Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Mecanismos de seguridad de la plataforma FIWARE

Autor: Javier Saavedra Galiano

Tutor: Juan Antonio Ternero Muíz

Departamento Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Mecanismos de seguridad de la plataforma FIWARE

Autor:
Javier Saavedra Galiano

Tutor:
Juan Antonio Ternero Muñiz
Profesor colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo de Fin de Grado: Mecanismos de seguridad de la plataforma FIWARE

Autor: Javier Saavedra Galiano

Tutor: Juan Antonio Ternero Muñiz

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Gracias, en especial a mi familia por tanto apoyo brindado en estos años de tanto esfuerzo y dedicación.

A mi padre, que gracias a sus charlas y consejos me han ayudado a lograr mi mayor objetivo.

A mi madre, que con su apoyo y cariño ha conseguido que cada día tuviera ilusión por ir a la Escuela.

A mis profesores, con los que he aprendido y disfrutado mucho de este maravilloso Grado, ya que de todos y cada uno de ellos he aprendido algo.

A mi novia, por aguantarme y seguir apoyándome a pesar de cualquiera de las circunstancias que se acontecieran.

Gracias por estos años tan especiales.

Javier Saavedra Galiano

Sevilla, 2019

Resumen

Uno de los campos de gran interés en la actualidad es la seguridad, concretamente la seguridad de los datos. Cada vez se maneja mayor cantidad de datos y se precisa de mecanismos de seguridad sólidos que se encarguen de frenar cualquier intento de acceso no deseado a los recursos que se encuentren en un sistema.

Este proyecto se ha centrado en la plataforma FIWARE, la cuál se encarga de proporcionar una serie de estándares para facilitar el desarrollo de *smart-solutions*[1]. Se han abordado los diferentes componentes que forman esta plataforma y la utilidad que tienen cada uno de ellos. Tras un estudio de los componentes que ofrece esta iniciatia, se han abordado en profundidad las diferentes herramientas que proporciona FIWARE para dotar de seguridad a aquellos sistemas basados en esta plataforma.

Este proyecto es una continuación del Trabajo Fin de Grado de Luis Martínez Ruiz desarrollado en 2018 [1], en el que se desarrolla un sistema formado por un sensor GPS y un giroscopio acelerómetro, conectados a una Raspberry Pi, la cuál se conectaba con Orion Context Broker. Tras analizar los diferentes mecanismos de seguridad que ofrece FIWARE, en el presente proyecto se ha basado en la ampliación del sistema original, incorporando nuevos componentes que dotan al sistema de mayor seguridad y usabilidad.

[1] Solución inteligente: Sistemas desarrollados con los componentes que ofrece FIWARE.

Abstract

One of the fields of great interest today is data security. More and more data is being handled and strong security mechanisms are required to stop any unwanted access attempts to resources in a system.

This project has focused on the FIWARE platform, which is responsible for providing standards to facilitate the development of smart-solutions. The different components that make up this platform and the usefulness of each of them have been addressed. After a study of the components offered by this initiative, the different tools that FIWARE offers to provide security to those systems based on this platform have been addressed in depth.

This project is a continuation of the Final Degree Project of Luis Martínez Ruiz developed in 2018, in which a system consisting of a GPS sensor and an accelerometer gyroscope is developed, connected to a Raspberry Pi, which was connected to Orion Context Broker. After analyzing the different security mechanisms offered by FIWARE, this project has been based on the extension of the original system, incorporating new components that give the system greater security and usability.

Índice

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Motivación

Este proyecto se ha desarrollado para analizar la actual situación de las aplicaciones del *IoT*[2] en lo que a seguridad se refiere. Se ha realizado un sistema basado en la plataforma FIWARE en el que se utiliza un sensor GPS para la recogida de datos, conectado a una Raspberry Pi para poder enviar la información recogida por el sensor al resto de componentes que forman nuestro sistema. En este proyecto se han incorporado nuevos elementos que mejoran los aspectos de seguridad como la autenticación y la gestión de permisos de nuestra smart-solution.

La plataforma FIWARE es una plataforma impulsada por la Unión Europea que empezó a desarrollarse en el año 2011. Esta plataforma es una iniciativa *Open Source*[3] cuyo objetivo es el desarrollo de aplicaciones y soluciones destinadas principalmente para la *Smart City*[4]. Al ser una plataforma Open Source, tiene una gran comunidad de desarrolladores que aportan sus conocimientos e ideas innovadoras a esta iniciativa[2].

En cualquier smart-solution es necesario recopilar y administrar información de contexto, procesar esa información e informar a los actores externos, lo que les permite actuar y, por lo tanto, modificar el contexto actual en base a la información recopilada. El componente FIWARE Context Broker es el componente principal de cualquier plataforma "Powered by FIWARE", el cual proporciona la *API* [5] NGSIv2. Esta API permite realizar actualizaciones, consultas o suscribirse a cambios en la información de nuestro sistema [3].

FIWARE proporciona componentes para mejorar la seguridad de las Smart-soluitions que utilicen su plataforma. El componente principal es Keyrock, el cuál se encarga de la gestión de la identidad basado en protocolos como OAuth2, lo que nos permite administrar la autenticación de usuarios y dispositivos, gestionar perfiles de los usuarios registrados en nuestro sistema y preservar la privacidad de los datos personales [4].

[2] Internet of Things: Concepto que comprende la conexión a internet de cualquier dispositivo cotidiano.
[3] Plataforma de código abierto: La licencia forma parte del dominio público.
[4] Ciudad inteligente: Ciudad que está dotada de sensores y actuadores, integradas en una Smart solution.
[5] Application Programming Interface: Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

1.2 Objetivos

El objetivo principal de este proyecto es analizar e implementar las herramientas de seguridad que proporciona la plataforma FIWARE para el desarrollo de Smart-solutions. Para ello se desarrollará un sistema que cuenta con un sensor GPS encargado de la recogida de datos, conectado a un agente IOT que se encarga de traducir los mensajes que envían los dispositivos IOT a mensajes en formato *NGSI*[6]. Al bloque anterior se le conecta el Context Broker, que se encarga de actualizar la información de contexto que envía el agente IOT y, finalmente, se incorporan las figuras encargadas de proveer seguridad a nuestro sistema. Por un lado, se incorpora el componente Keyrock, que se encargará de la gestión de identidad de los usuarios que formen parte de nuestro sistema. En segundo lugar, se añade el proxy *PEP*[7] Wilma, encargado de permitir o denegar las peticiones procedentes de las aplicaciones en función de los permisos que tenga el usuario y la aplicación que los envíe. Para la comprobación del correcto funcionamiento del sistema, se utilizará una aplicación web sin interfaz gráfica.

1.3 Antecedentes

El punto de partida de este proyecto es el Trabajo de Fin de Grado de Luis Martínez Ruiz [2], realizado en el año 2018. En dicho proyecto se desarrolla un sistema formado por los siguientes componentes: Un sensor GPS, un giroscopio/acelerómetro, una Raspberry Pi, un Context Broker (Orion) y finalmente un dashboard (Freeboard) para la representación de los datos recogidos por los dispositivos IOT del sistema.

En el presente proyecto, se ha hecho una ampliación del sistema indicado anteriormente y se han llevado a cabo una serie de mejoras respecto a la versión anterior del mismo. En primer lugar, se ha incorporado la figura del agente IOT, el cuál facilita la comunicación entre el Context Broker y los dispositivos IOT. En nuestro caso el dispositivo IOT está formado por un sensor GPS, cuyos datos transmite la Raspberry Pi. En segundo lugar, se añaden los componentes Keyrock y Wilma, encargados de proporcionar herramientas de seguridad en el acceso a los datos. Finalmente, se ha prescindido de la figura del *dashboard*[8] para la representación de los datos, ya que este no es compatible con todas las funcionalidades que proporciona Keyrock. En lugar de un dashboard, se ha elegido el software Postman para realizar peticiones HTTP a las diferentes APIs que proporcionan componentes del sistema.

1.4 Descripción de la solución

En este apartado se explicará los componentes que forman el sistema y se detallarán las funciones de cada uno de ellos.

[6] API Rest desarrollada por Fiware que permite la comunicación entre los diferentes componentes de Fiware y Orion Context Broker.
[7] Policy Execution Point: Punta de ejecución de Políticas
[8] Aplicación que permite la visualización mediante widgets de los datos.

1.4.1 Arquitectura del Sistema

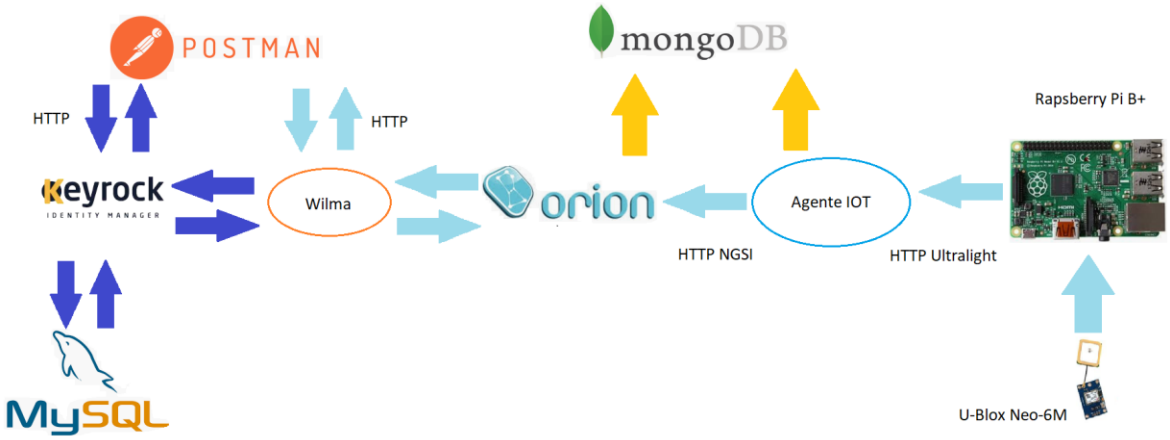


Ilustración 1: Arquitectura del Sistema

1.4.2 Funcionalidades

En esta sección se procede a explicar de forma general los diferentes componentes que conforman el sistema. Para ello, se va a dividir el sistema en cuatro bloques funcionales.

- Bloque 1: Este bloque lo forman el sensor GPS y la Raspberry Pi B+. Su función es la de recoger información de interés a partir del sensor para su posterior procesamiento y tratamiento. Esta información se envía a la Raspberry Pi, la cual recupera esta información mediante un programa desarrollado en el lenguaje Python.
- Bloque 2: Este bloque lo forman el agente IOT, Orion Context Broker y MongoDB. La función de este bloque es la más compleja, ya que es la pieza central de nuestro sistema. El agente IOT procesa la información que recibe de la Rapsberry Pi para que Orion pueda almacenar los datos de interés en la base de datos MongoDB. El agente IOT recibe mensajes en formato Ultralight de la Raspberry y los traduce al mensajes en formato NGSI para que Orion pueda almacenar esa información de contexto en MongoDB.
- Bloque 3. Este bloque lo forman Wilma PEP Proxy, Keyrock y MySQL. Estos componentes son los encargados proveer diversos mecanismos de seguridad, tales como autenticación y gestión de permisos a partir de unos roles determinados. Keyrock es la pieza central de este bloque, ya que es el encargado de decidir si permite o deniega el acceso por parte de un usuario o aplicación a cierta información. En nuestro sistema Keyrock ocupa la figura de *PDP*[9]. El siguiente componente es el proxy PEP Wilma, cuya función es permitir o denegar el acceso a un recurso del sistema en función de la decisión que tome el PDP. Para la gestión de la autenticación se utiliza una base de datos Mysql, con la cuál se comunica Keyrock para tomar las decisiones.
- Bloque 4: Este bloque lo conforma únicamente el programa Postman. Este programa será el encargado de realizar todas las tareas dentro de nuestro sistema, ya que carecemos de aplicaciones web con interfaz gráfica o dashboards compatibles con las funcionalidades que nos proporciona Keyrock. Estas tareas se llevarán a cabo a través de peticiones HTTP a las diferentes APIs del sistema en función de la tarea que se quiera realizar.

[9] Policy Decision Point: Punto de decisión de Políticas.

1.5 Estructura de la memoria

En este apartado se proporciona a modo de guía, un breve resumen de los apartados que componen la memoria:

1. **Introducción:** Explicación del sistema desarrollado y de la necesidad del mismo.
2. **Recursos utilizados:** Muestra los recursos empleados para el desarrollo de la smart-solution. Esto incluye los recursos software y hardware necesarios para el correcto funcionamiento del Sistema.
3. **Tecnologías utilizadas:** Introducción de las diferentes tecnologías utilizadas en el Sistema.
4. **Plataforma FIWARE:** Explicación amplia de FIWARE en la que se detallan los diferentes componentes que lo forman.
5. **Mecanismos de seguridad en FIWARE.** Estudio de los diferentes mecanismos de seguridad que ofrece FIWARE y su integración en el sistema.
6. **Smart-Solution desarrollada:** Explicación de cada uno de los componentes del Sistema desarrollado y sus funcionalidades.
7. **Conclusiones y líneas futuras:** Tras el Desarrollo del Proyecto, se exponen las diferentes concluiiones obtenidas e ideas para futuras aplicaciones que tengan como punto de partida el presente Proyecto.

Finalmente se ofrece una serie de anexos a modo de manual para poder hacer uso del Sistema desarorollado:

- **Anexo A:** Configuración de Rapsberry Pi y sensor GPS.
- **Anexo B:** Instalación de los componentes del sistema.

2 RECURSOS UTILIZADOS

En este capítulo se tratarán los diferentes recursos que se han empleado para la realización de este proyecto. Para ello, diferenciaremos dos tipos de recursos: Recursos hardware y recursos software.

2.1 Recursos Hardware

En esta sección se mostrarán aquellos recursos hardware que se han utilizado para llevar a cabo este Proyecto.

2.1.1 Sensor U-Blox NEO-6M

Este sensor será el encargado de recoger los datos GPS de interés. En este caso recogeremos los datos de latitud, longitud, altitud, velocidad, dirección, subida y fecha.

Ilustración 2: Sensor U-Blox NEO-6M

Las principales características son las siguientes [5]:

- Voltaje de alimentación: 3-5 VDC.
- Interfaz: Serial UART 5V.
- Antena cerámica.
- Indicador de señal con LED

La comunicación con la Raspberry Pi se establece mediante *UART*[10].

[10] Universal Asynchronous Receiver-Transmitter: Transmisor-Receptor Asíncrono Universal. Protocolo de comunicación entre componentes.

2.1.2 Raspberry Pi Model B+

Este componente es el encargado de recoger los datos del sensor y enviarlos en un formato correcto al agente IOT. Para conectar la Raspberry Pi con el resto de componentes se ha utilizado un dispositivo Wireless USB Network. Esto permite conectarnos a internet vía Wi-Fi para poder enviar la información que se desee. Finalmente, para poder configurar y utilizar la Raspberry Pi se ha instalado el sistema operativo Raspbian, basado en Debian.

Ilustración 3: Raspberry Pi Model B+

Las características de la Raspberry Pi B+ son [6]:

- *SoC*[11]: Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)
- Procesador: CPU ARM1176JZF-S (ARMv6) a 700 MHz, con aritmérica de punto flotante, arquitectura de 32 bits
- *GPU*[12]: VideoCore IV. Soporta OpenGL ES 2.0, MPEG-2 y VC-1.
- Memoria RAM: RAM 512 MB compartida con la memoria de video de la GPU.
- Salida de audio por conector estándar de 3.5 mm y por HDMI.
- Ranura para microSD para almacenar el sistema operativo y programas.
- Ethernet 10/100 con conector RJ45
- Entrada de video con conector MIPI CSI
- 4 puertos USB 2.0 (Conectores tipo A hembra).
- 28 pines entrada/salida de uso general (GPIO) con UART, Bus I2C y Bus SPI
- Conector USB micro B hembra para alimentación.

2.1.3 Ordenador portatil ASUS F550-CA

En este ordenador es donde se ha desarrollado el grueso del Proyecto. En él se encuentran los siguientes componentes:

- Agente IOT.
- Orion Context Broker.

[11] System on a chip: Sistema en un chip formado por CPU + GPU + DSP + SDRAM + USB
[12] Graphics Processing unit: Unidad de Procesamiento Gráfico

- Mongo DB.
- Proxy PEP Wilma.
- Keyrock.
- MySQL.
- Postman.

Ilustración 4: Ordenador portátil ASUS

Las características más importantes del ordenador son las siguientes[8]:

- Procesador: Intel Core i7-3537-U a 2.0 GHz.
- Memoria RAM: DDR3L de 8GB.
- Disco duro: SSD Kingston 480GB.
- Tarjeta Gráfica: Intel HD Graphics 4000
- Sistema Operativo: Ubuntu 16.04 LTS

2.2 Recursos Software

En esta sección se mostrarán aquellos recursos hardware que se han utilizado para llevar a cabo este Proyecto.

2.2.1 VMware Workstation 14 Player y Ubuntu 16.04 LTS

Ilustración 6: Logo Ubuntu

Ilustración 5 - Logo VMware

En el Desarrollo de este Proyecto se ha utilizado el Sistema operativo Ubuntu 16.04 LTS. Este Sistema Operativo se ha utilizado como una máquina virtual creada en VMware Workstation 14 Player. La decision de utilizar Ubuntu como Sistema operativo ha sido el uso la tecnología Docker, la cuál está perfectamente implementada en este Sistema Operativo.

2.2.2 Postman

Ilustración 7: Logo Postman

Este proyecto carece de aplicación web con interfaz gráfica o de un dashboard con el que sea intuitivo actuar. Por ello, el programa que se utiliza para interactuar con el sistema es Postman. Este programa permite realizar diversas tareas dentro del mundo API Rest, como crear peticiones a APIs internas o de terceros [9]. Esta tarea es la que se usará en este proyecto, la de crear peticiones para comunicarnos con los diversos componentes del sistema gracias a peticiones HTTP en formato JSON a las diferentes APIs que proporciona cada componente.

Para simplificar la explicación de las peticiones que se realizan a través de Postman, se utilizará una herramienta que proporciona este software que permite obtener comandos curl a partir de las peticiones que se configuren en la interfaz gráfica.

2.2.2.1 Caso de uso en Postman

En este apartado se hará uso del software Postman en un caso de uso concreto para mostrar las funcionalidades que se han usado en este proyecto.

Se muestra una interfaz gráfica en la que se permite configurar los diferentes parámetros de una petición HTTP.

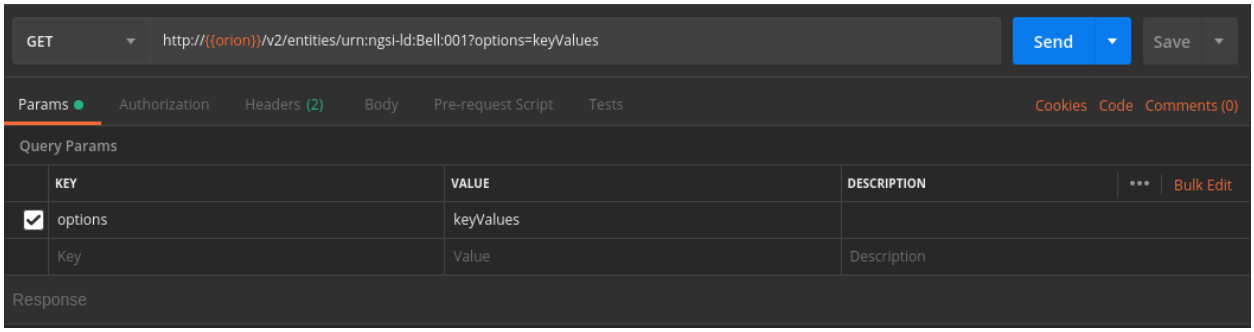


Ilustración 8: Interfaz gráfica Postman

En primer lugar, se configura el tipo de petición HTTP. En este proyecto se han hecho uso de las peticiones HTTP GET y HTTP POST.

En segundo lugar, se configura la URL a la que se hará la petición. En el caso de la Ilustración 8, la URL a la que se accederá será *http://{{orion}}/v2/entities/urn:ngsi-ld:Bell:001* .

Finalmente se procede a configurar de la petición, para ello se han utilizado las siguientes herramientas:

- Params: Parámetros de la petición HTTP. En el caso de la Ilustración 8, se asigna el valor "keyValues" al campo "options". Esto se concatena al final de la URL y permite configurar cómo se hará la respuesta por parte del servidor.
- Authorization: Permite configurar la cabecera de Autorización de la petición. En este proyecto no se ha utilizado esta herramienta.
- Headers: Se especifican las diferentes cabeceras de la petición HTTP. En este caso se han especificado 2 cabeceras.
- Body: El cuerpo del mensaje. Este parámetro es empleado en las peticiones POST que se han utilizado en este proyecto.

Una vez se ha configurado convenientemente la petición HTTP con la interfaz gráfica de Postman, procedemos a obtener el comando Curl correspondiente a dicha configuración. Esto se hace seleccionando el botón "*Code*" visible en la parte derecha de la Ilustración 8.

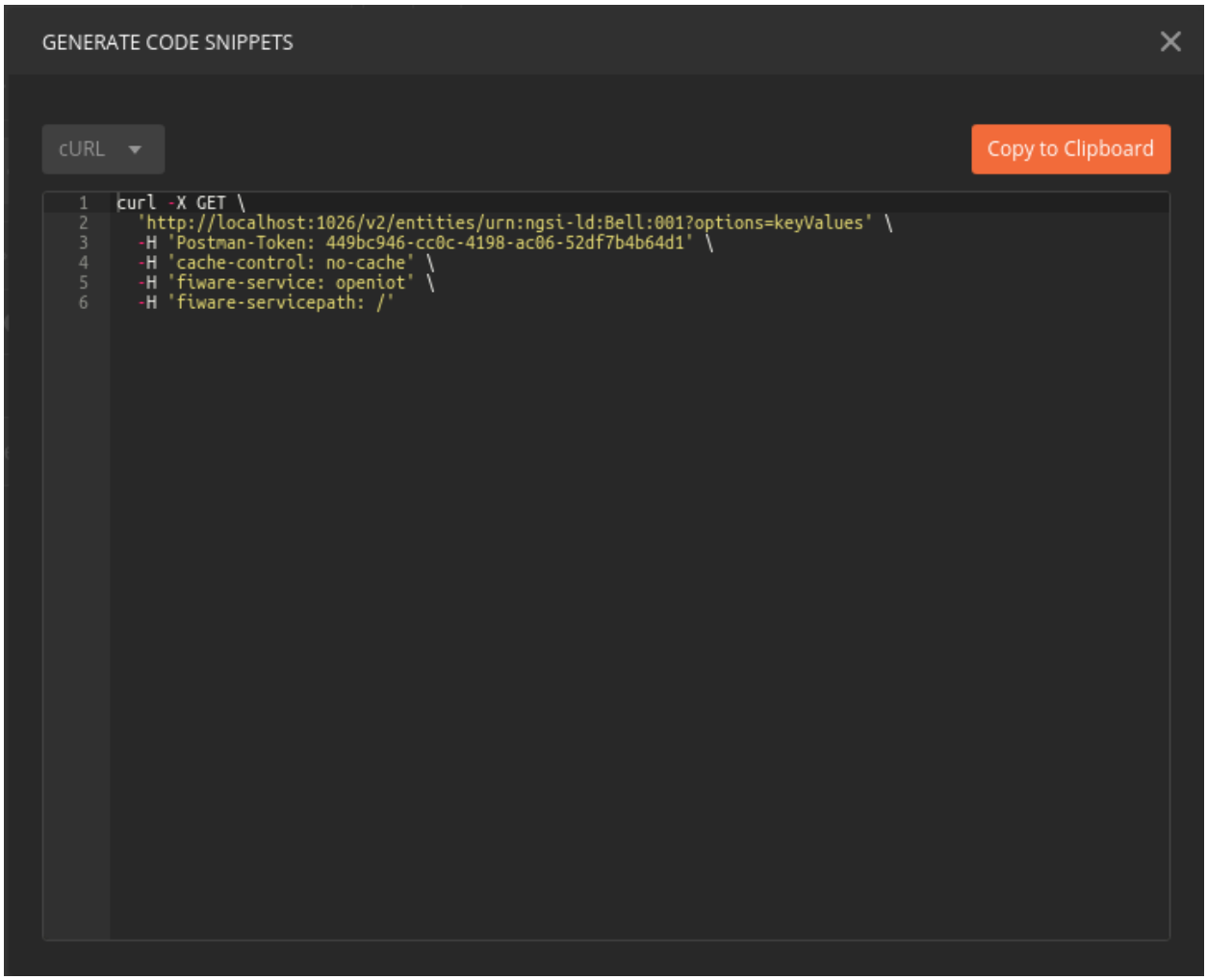


Ilustración 9: Comando Curl en Postman

Una vez se accede a la ventana de la Ilustración 9, se elige en la esquina superior izquierda el formato que deseamos del código, que en nuestro caso sera Curl.

3 TECNOLOGÍAS UTILIZADAS

Se procede a detallar en este capítulo cada una de las tecnologías empleadas para el desarrollo de la Smart-solution. Cabe destacar que se comentará todo tipo de tecnologías empleadas, desde lenguajes de programación a las diferentes herramientas utilizadas.

3.1 Python

Ilustración 10: Logo Python

Python es el lenguaje de programación que se ha usado para interpretar los datos recogidos por el sensor GPS y por el giroscopio/acelerómetro. También se ha empleado esta tecnología para enviar los datos de los sensores desde la Rapsberry Pi hasta el agente IOT, mediante peticiones HTTP POST [10].

La decision de utilizar esta tecnología se debe a que Python dispone de librerías como gpsd que recogen los datos de los sensores conectados a los puertos de la Raspberry Pi.

3.2 Docker y Docker-compose

Docker es la tecnología elegida para alojar el grueso de los componentes que forman el sistema. Esta tecnología utiliza el kernel de Linux y las herramientas de este para separar los procesos, lo cual permite la ejecución de cada uno de los mismos de forma independiente. Cada uno de estos procesos están ejecutándose en contenedores independientes unos de otros, lo cuál conserva la seguridad de estos procesos al funcionar como sistemas separados [11].

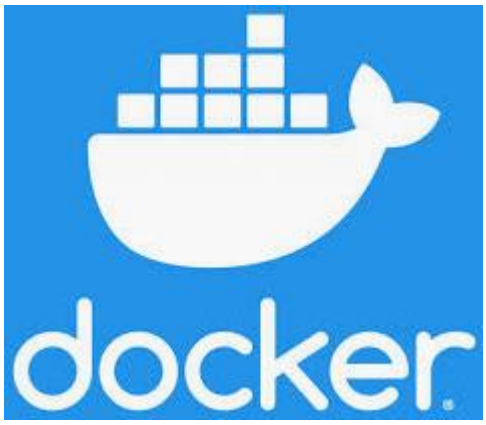


Ilustración 12: Logo Docker

Ilustración 11: Logo Docker-compose

Esta tecnología ofrece un modelo basado en imágenes, es decir, permite obtener una aplicación o servicio junto a todas y cada una de las dependencias necesarias para poder usarlas.

Para simplificar el uso de Docker se ha utilizado la herramienta Docker-compose, la cuál permite generar scripts que facilitan el diseño y la construcción de estos servicios [12]. Con esta herramienta se pueden definir y ejecutar aplicaciones Docker multi-contenedor, como es el caso de este proyecto. Se detallará la configuración y uso de Docker y Docker-compose en el anexo B.

3.3 Orion Context Broker

Orion Context Broker es la piedra angular del Sistema. Se trata de una implementación en C++ de la API REST NGSIv2.

Ilustración 13: Logo Orion Context Broker

Orion Context Broker permite gestionar la información de contexto mediante actualizaciones, consultas, registros y subscripciones [13]. Esto es posible gracias a elementos de contexto como las entidades, en las cuales se almacena la información de contexto para su posterior tratamiento. Se entrará más en detalle en el Capítulo 4.

3.4 Keyrock y Wilma PEP Proxy

Keyrock ocupa la figura de *Identity Manager*[13] dentro del sistema. La gestión de identidad concede derechos individuales para acceder a determinados recursos en un momento concreto y por unas razones determinadas [4].

Ilustración 14: Logo Keyrock

Esta herramienta de FIWARE proporciona una serie de componentes separados encargados de comprobar quíen o qué tiene acceso a cada uno de los recursos dentro del sistema, todo ello conociendo de antemano la identidad de la figura que realice la petición de acceso a cada recurso.

Keyrock es el PDP del sistema, es decir, se encarga de tomar la decisión de permitir o denegar las peticiones que hacen los componentes del sistema , ya sea una aplicación o un usuario, en base a los permisos que estos tengan. A este componente se le añade la figura del proxy PEP Wilma [14], cuya función de es de ejercer la decisión que tome Keyrock respecto a las peticiones que se produzcan. Se entrará más en detalle en el Capítulo 5.

3.5 MongoDB y MySQL

MongoDB y MySQL son los sistemas de base de datos utilizados en el sistema. MongoDB se usa para almacenar la información de contexto que registran tanto Orion Context Broker como el agente IOT. MySQL es el encargado de almacenar la información de autenticación y permisos que gestiona Keyrock.

Ilustración 16: Logo mongoDB

Ilustración 15: Logo MySQL

MongoDB es un sistema de base de datos *NoSQL*[14] que almacena la información en estructuras de datos *BSON*[15]. Es una base de datos orientada a documentos, ya que en vez de almacenar la información en tablas y registros como se hacía tradicionalmente, se almacenan los datos en documentos [15].

MySQL es un sistema de base de datos relacional, es decir, utiliza tablas que se relacionan entre sí para almacenar la información y seguir una organización eficiente. Para el acceso y manejo de los datos en MySQL se utiliza el lenguaje SQL [16].

[13] Gestor de identidad: Es la entidad responsable de la gestión de la identidad en un sistema.
[14] Son aquellos sistemas de bases de datos que no utilizan SQL para su gestión.
[15] Consiste en un formato de intercambio de datos basado en el término JSON (JSON Binario)

4 PLATAFORMA FIWARE

4.1 Introducción a FIWARE

FIWARE es una iniciativa impulsada por la Unión Europea cuya finalidad es favorever el desarrollo de aplicaciones y soluciones principalmente vinculadas a las Smart Cities. Esta iniciativa Open Source se desarrolló en el año 2011 y cuenta con una amplia comunidad de desarrolladores que siguen aportando ideas para el continuo crecimiento de esta plataforma. También cuenta con el respaldo de importantes empresas como Atos, Telefónica y NEC entre otros [17].

Esta plataforma tiene como principal objetivo construir un ecosistema abierto basado en estándares de dominio público y en herramientas que faciliten el desarrollo de nuevas aplicaciones inteligentes, capaces de adaptarse a cualquier sector. Este proyecto se ha podido desarrollar gracias a la extensa documentación que aporta esta plataforma a los desarroladores.

Estos estándares reciben el nombre de *Generic Enablers*[16]. FIWARE proporciona una serie de Generic Enablers para facilitar el desarrollo de aplicaciones y ahorrarles tiempo de desarrollo a aquellos que decidan realizar una Smart-Solution. Estos estándares son desarrollados por empresas de terceros y proporcionan APIIs cuyas especificaciones son públicas. Para que una herramienta reciba el nombre de FIWARE Generic Enabler, el desarrollador debe solicitar a FIWARE que dicha herramienta forme parte de su catalogo, cumpliendo previamente una serie de condiciones. Para desarrollar una smart-solution no es necesario el uso de todos y cada uno de los Generic Enablers que existen, el único componente indispensable es el Context Broker. El resto de Generic Enablers sirven como complemento al Context Broker, y se discutirá la utilidad de estos en función de la aplicación a desarrollar[18].

Las Smart-solutions tienen como principal característica la recogida de datos de contexto desde dispositivos IOT. Estos dispositivos pueden ser sensores, aplicaciones móviles etc. Esta información se recopila y se administra para tomar decisiones en base a los datos obtenidos.

Ilustración 17: Esquema básico de una Smart-Solution

[16] Componente estándar que proporciona la plataforma Fiware.

4.2 FIWARE Generic Enablers

En esta sección se procede a explicar los diferentes Generic Enablers que proporciona FIWARE y las funcionalidades que aportan los mismos.

4.2.1 Core Context Management

4.2.1.1 Orion Context Broker

El componente principal de cualquier Smart-solution "Powered by FIWARE" es Orion Context Broker. Este estándar proporciona mecanismos de administración, actualización, registro y subscripción a la información de contexto del sistema [19].

Orion organiza la información de contexto en entidades, de forma que estas se puedan relacionar entre sí fácilmente. Estas entidades representan información del mundo real y son accedidas desde los diferentes componentes del sistema donde cada uno de ellos realiza una acción sobre la información de contexto almacenada. Los campos imprescindibles en toda entidades son:

- Id: Identificador único en el sistema que identifica unívocamente a la entidad.
- Type: Identifica el tipo de la entidad. Este campo se utiliza para identificar el objeto del mundo real en el sistema.
- Atributos: Muestran los datos del mundo real de la entidad en cuestión. Estos atributos dependerán del tipo de la entidad. Por ejemplo, la entidad “termómetro” tendrá un atributo de nombre “temperatura”.

```
curl -X POST \
  http://localhost:1026/v2/entities/ \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: be29b61a-f48a-4d9f-80cc-347df548b341' \
  -H 'cache-control: no-cache' \
  -d '{
    "id": "urn:ngsi-ld:Store:001",
    "type": "Store",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Antonio Montes 17",
            "addressRegion": "Alcalá de Guadaira",
            "addressLocality": "Sevilla",
            "postalCode": "41500"
        },
        "metadata": {
            "verified": {
                "value": true,
                "type": "Boolean"
            }
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [37.336702, -5.845826]
        }
    },
    "name": {
        "type": "Text",
        "value": "Alcalá de Guadaira"
    }
}'
```

Ilustración 18: Entidad que muestra datos sobre una tienda

La principal carácteristica de Orion Context Broker es que implementa la API NGSIv2, la cual permite actualizar, consultar y subscribirnos a la información de contexto, la cuál se encuentra almacenada en entidades.

NGSI es una API Rest que permite la comunicación mediante mensajes HTTP entre los diferentes componentes del sistema. Con estos mensajes HTTP se accede a la información de las entidades, ya sea modificándola, consultándola, etc.

Los datos se recogen en la capa de gestión por sensores, robots o sistemas de terceros y se almacenan temporalmente en el Context Broker, es decir, Orion solo almacena el último dato recibido por la capa de adquisición de datos. Con el objetivo proporcionar una capa de persistencia a los datos que maneja Orion, este se comunica con un gestor de base de datos para almacenarlos [20].

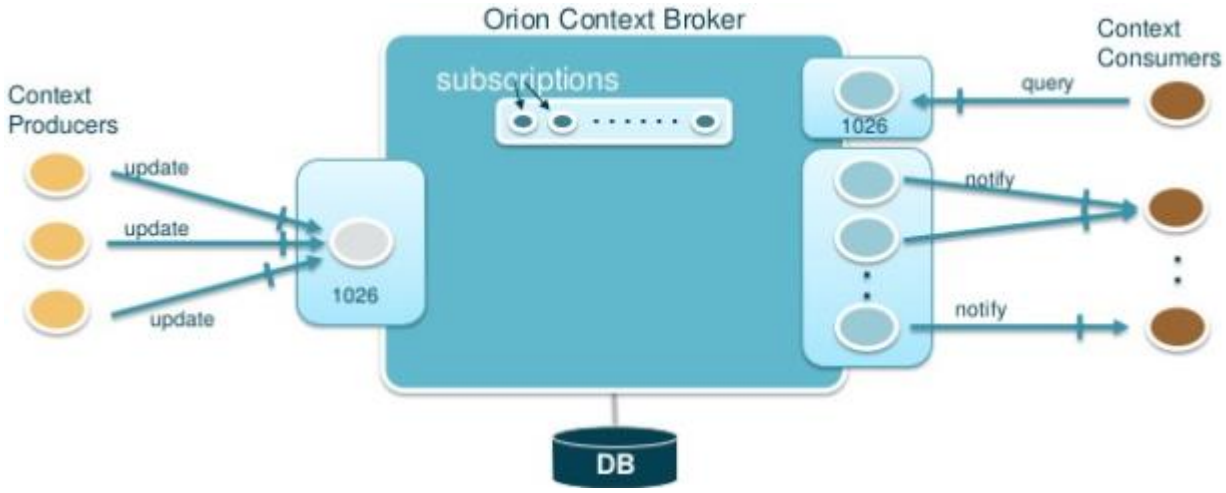


Ilustración 19: Comunicación con el Context Broker

Existen dos tipos de componentes que actúan sobre el Context Broker:

- Context Producers: Aquellos dispositivos IOT capaces de recoger información y enviarsela al context broker en el formato correcto.
- Context Consumers: Son aquellos que interactúan con el context broker para consultar o modificar la información de contexto almacenada. Lo normal es que esta figura sea una aplicación que haga peticiones a la API NGSI que implementa el context broker.

4.2.1.2 STH Comet

STH[17] Comet es un componente del ecosistema FIWARE encargado de almacenar y recuperar la información histórica de los datos de contexto registrados de una instancia de Orion Context Broker. Todas las comunicaciones que se establecen con Comet están estandarizadas por la API NGSI [21].

Con este componente se puede analizar el comportamiento de una entidad observando los datos *brutos*[18] históricos almacenados en Orion Context Broker en un intervalo de tiempo.

Para su correcto funcionamiento, Comet se basa en los siguientes conceptos:

[17] Short Time Historic: Histórico de tiempo limitado.
[18] Estado en que se encuentra la información al obtenerse del mundo real.

- Resolución: Unidad de tiempo en el cual se agrupa la información. Este periodo puede establecerse en mes, día, hora, minuto y segundo.
- Origen: Se establece un punto de partida en el tiempo en función de la resolución escogida. Se puede establecer cualquier unidad al igual que la resolución.
- Offset: Es el desplazamiento respecto del origen en el que se almacena la información de contexto.
- Muestras: Para una resolución, origen y offsets concretos, se determinan el número de muestras, valores, eventos y notificaciones disponibles para un concreto desplazamiento respecto del origen.

Este componentes se integra con Cygnus Generic Enabler, el cuál se ve en el siguiente apartado.

4.2.1.3 Cygnus

Este proyecto forma parte de la plataforma FIWARE, formando parte también del ecosistema *COSMOS*[19].

Cygnus es un componente encargado de conectar la información de contexto almacenada en Orion Context Broker con un sistema de almacenamiento desarrollado por terceros como STH Comet, otorgando asi una visión histórica de los datos [22].

Internamente Cygnus está basado en Apache Flume[23], que es un servicio distribuido que almacena y distribuye de forma fiable y eficiente grandes cantidades de datos. Su arquitectura es sencilla y flexible, basada en flujos de datos en *streaming*[20], lo cuál permite construir múltiples flujos a través de diferentes agentes hasta que alcanzan el destino final. Estos agentes están compuestos por un elemento encargado de recibir los datos para posteriormente transmitirlos, un canal a travez del cuál se transportan los datos y, finalmente, un sumidero que recibe los datos para almacenarlos en el sistema de almacenamiento de terceros que se use.

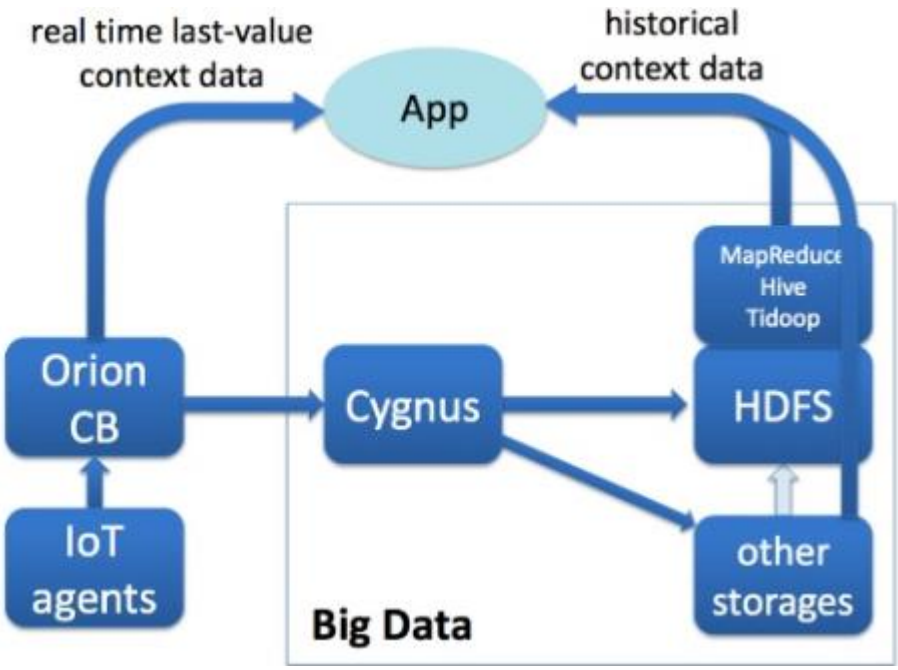


Ilustración 20: Arquitectura FIWARE usando Cygnus

[19] Generic Enabler dedicado a operaciones de Big Data
[20] Emisión continua de datos.

Cygnus está diseñado para ejecutar un agente Flume específico por cada fuente de datos. Su utilidad principal es la de gestionar grandes cantidades de datos (Big Data).

4.2.2 Interfaz con IOT, Robots y Sistemas de Terceros

Existen variedad de Generic Enablers cuya función es la de facilitar el acceso a los dispositivos IOT y traducir tanto la información que estos transmiten como las actuaciones que como usuarios queremos hacer sobre estos dispositivos.

El componente encargado de realizar dichas acciones es el agente IOT. Estos agentes IOT tienen un puerto norte y un puerto sur. El puerto norte es el encargado de establecer la comunicación con Orion Context Broker, de forma que se puedan enviar y recibir mensajes en el formato adecuado. Este puerto también es el que se usa para configurar el agente IOT. En el puerto sur se reciben los mensajes generados por los dispositivos IOT. Cada dispositivo IOT empleará un protocolo determinado, el cuál debe ser conocido por el agente IOT para llevar a cabo su posterior traducción al formato NGSI.

El agente IOT debe ser configurado previamente para que este sea capaz de entender los mensajes que les envían los dispositivos. Por ejemplo, si una entidad tiene el atributo “temperatura”, debemos asociar ese atributo a la clave “t”, para que cuando el dispositivo en cuestión envíe un valor asociado a esa clave, el agente IOT pueda traducir dicho mensaje en una petición NGSI.

Se procede a detallar los agentes IOT más usados.

4.2.2.1 Agente IOT JSON

Un agente IOT JSON es un puente que se usa para comunicar dispositivos IOT usando el protocolo JSON y un Context Broker basado en NGSI como Orion Context Broker [24].

JSON es un formato de texto sencillo para el intercambio de datos. Recibe este nombre ya que es un formato fácil de leer y escribir para el humano y fácil de interpretar y generar para las máquinas.

JSON está constituído por dos estructuras:

- Una colección de pares nombre/valor.
- Una lista ordenada de valores.

Este agente IOT traduce los mensajes que envían los dispositivos en formato *MQTT*[21]a peticiones NGSI en el formato adecuado.

[21] Message Queuing Telemetry Transport.

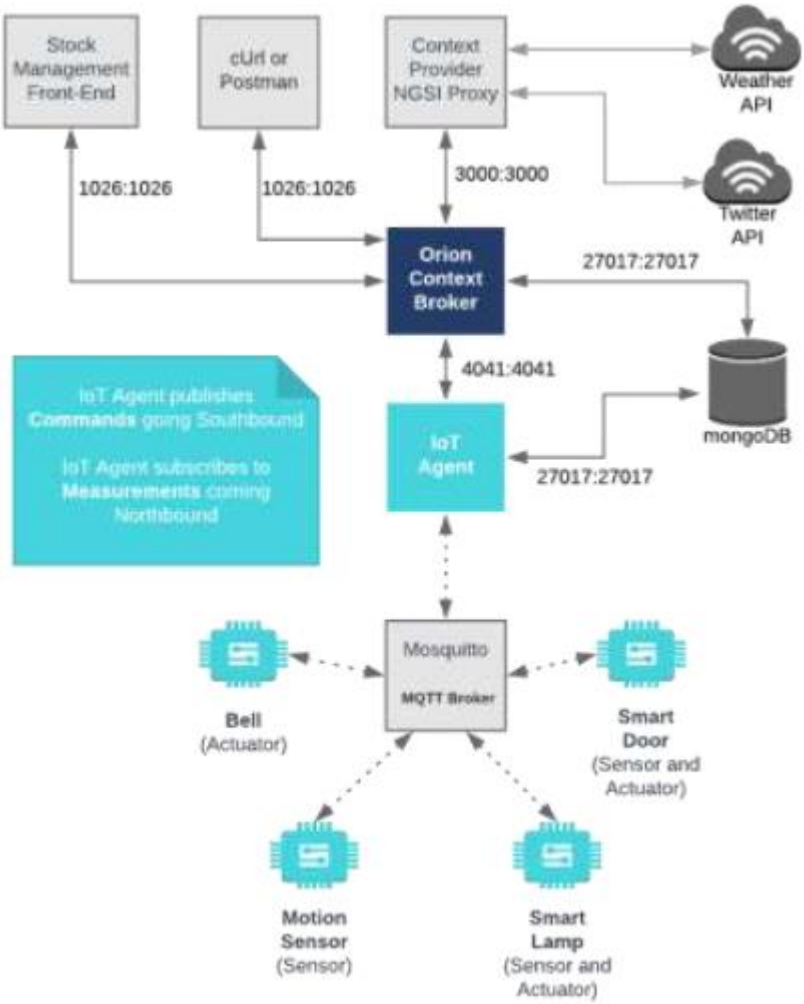


Ilustración 21: Arquitectura de un sistema Fiware usando MQTT

El protocolo MQTT fue diseñado para ejercer un transporte de mensajes de publicación/subscripción super ligero. Es útil en situaciones en las que se envían poca cantidad de información y se dispone de un ancho de banda limitado [25].

La pieza central para poder establecer una comunicación entre el emisor y el receptor es el "topic". A partir de un "topic" común para el emisor y el receptor se puede llevar a cabo esta comunicación.

La arquitectura del sistema sigue una topología en estrella, donde el nodo central es un broker. En FIWARE disponemos del MQTT Broker Mosquitto, el cuál se encarga de gestionar la red, trasmitir los mensajes y asignar los topics.

El formato de los mensajes MQTT es sigue un formato clave/valor de la siguiente forma:

```
'{"l":4,"t": "31.5"}'
```

Ilustración 22: Formato de mensajes usando el protocolo MQTT

4.2.2.2 Agente IOT Ultralight

Este agente IOT se encarga de traducir los mensajes transmitidos en formato HTTP Ultralight al formato HTTP NGSI.

Ultralight es un protocolo simple basado en texto que permite a los dispositivos IOT enviar datos y recibir comandos de forma sencilla. Este protocolo fue definido por la comunidad FIWARE con el propósito de establecer un protocolo sencillo de comunicación para dispositivos IOT [26].

El cuerpo de los mensajes que utilizan el protocolo Ultralight siguen el formato clave/valor de la siguiente forma:

Ilustración 23: Formato de mensajes usando el protocolo Ultralight

Como se observa en la Ilustración 21, los atributos se separan con el carácter '|'. Con el protocolo Ultralight no es necesaria la figura del "Broker", a diferencia del protocolo MQTT.

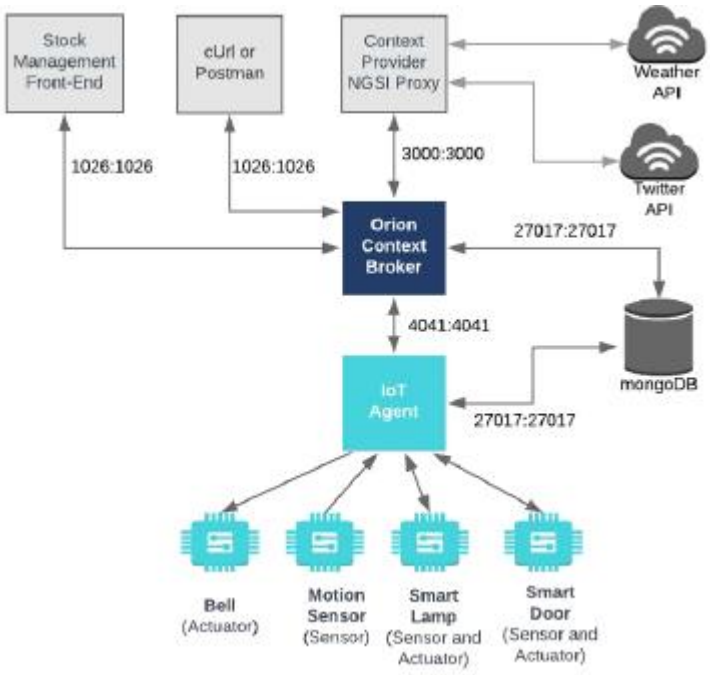


Ilustración 24: Arquitectura de un sistema Fiware usando Ultralight

4.2.3 Procesamiento, análisis y visualización de la información de contexto

FIWARE también proporciona multitud de Generic Enablers que se encargan de aquellos elementos que tienen una interacción directa con el usuario.

En primer lugar, nos ofrece herramientas para el procesado de datos, partiendo de unos datos brutos y obteniendo como resultado información de interés.

En segundo lugar, se dispone de diferentes plataformas que permiten la visualización a partir de dashboards totalmente configurables para poder representar los datos como requiera el usuario.

Finalmente, se proporcionan una serie de herramientas que facilitan el análisis Big Data de la información de contexto recogida por los dispositivos IOT y facilitando la integración de estos con las plataformas Big Data más populares.

4.2.3.1 Wirecloud

Wirecloud es una plataforma de mashup de aplicaciones web en la que el usuario desarrolla su propio entorno de visualización de datos sin necesidad alguna de tener conocimientos de programación. Otra funcionalidad que ofrece Wirecloud es la de controlar los dispositivos conectados desde este mashup creado por el propio usuario[27].

Un mashup de aplicaciones está formado por diferentes componentes, cada uno de ellos con una funcionalidad. De forma manual, se puede crear un workspace donde realizar el mashup y, a partir de ese workspace, situar manualmente cada uno de los componentes que formen el mashup en la disposición que desee el usuario.

Existen componentes de visualización, llamados Widgets, en los que simplemente se representan los datos recogidos por las fuentes de datos. Existen multitud de visualizadores que se adaptan a la necesidad de cada smart-solution. Por ejemplo, si una smart-solution dispone de un sensor GPS, existe la opción de crear un componente "Visor de mapa", en el que es posible indicar la posición en el mapa en base a las coordenadas recogidas por el sensor GPS.

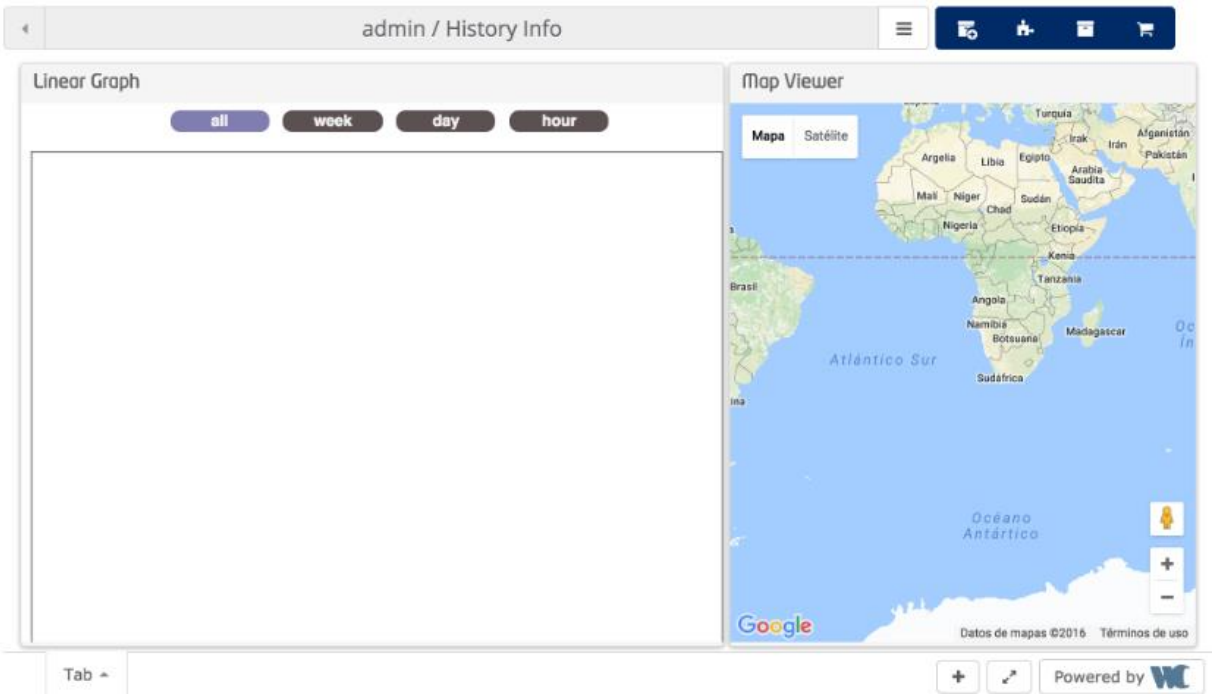


Ilustración 25: Ejemplo de mashup de aplicaciones usando Wirecloud

Otro componente importante en Wirecloud son los operadores, ya que son los intermediarios entre los Widgets y las fuentes de datos del sistema. Estas fuentes pueden ser tanto internas como externas. Los operadores son los encargados de proporcionar información a los widgets, que a su vez proviene de los dispositivos IOT o de servicios de sistemas de terceros.

Con la combinación de los diferentes widget y operadores existentes en Wirecloud, se puede llegar a crear aplicaciones Web muy pontentes, simplemente conectándolos de manera gráfica y sin necesidad de intervenir programáticamente.

4.2.3.2 Knowage

Knowage es una suite open source que proporciona herramientas para el análisis Big Data de negocios. Este conjunto de herramientas está dividido en módulos, cada uno dedicado a un dominio de análisis específico. Estos módulos pueden usarse de forma individual para desarrollar una solución completa, o bien se puede combinar entre ellos para conseguir cumplir los requisitos del usuario final [28].

Los sub-productos que ofrece Knwoage son:

- SI (Smart Intelligence): Sirve para llevar a cabo la inteligencia empresarial traducional a partir de datos estructurados.
- ER (Enterprise Reporting): Para producir y distribuir informes empresariales.
- LI (Local Intelligence): Para relacionar datos comercioales con información geográfica.
- PM (Performance Management) : Sirve para gestionar *KPIs*[22] y organizar cuadros de mando.
- PA (Predictive analysis): Para proporcionar más análisis avanzados para predecir el comportamiento futuro.
- EI (Embedded Intelligence): Permite vincular Knowage con soluciones externas que proporcione el cliente o están desarrolladas por terceros.

4.2.3.3 Kurento

Kurento es el Generic Enabler orientado a la transmisión, el cuál proporciona una serie de herramientas dedicadas a simplificar el desarrollo de aplicaciones multimedia complejas a través de una API.

Este componente proporciona un servidor multimedia y un conjunto de clientes de API para hacer más sencillo el desarrollo de apliaciones web y aplicaciones móviles [29].

Las principales características de Kurento son:

- Soporte de comunicación en grupo.
- Transcodificación de flujos audiovisuales.
- Grabación de flujos audiovisuales.
- Mezcla de flujos audiovisuales.
- Difusión de flujos audiovisuales.
- Enrutamiento de flujos audiovisuales.
- Procesamiento de medios que permiten la visión por ordenador.
- Realidad aumentada.

[22] Indicadores clave de rendimiento. Empleado en el ámbito empresarial.

- Análisis del habla.

La integración de todas estas características son posibles gracias a la arquitectura modular de este Generic Enabler. Gracias a ello se integran multitud de algoritmos de procesamiento de terceros que los desarrolladores de aplicaciones pueden usar de forma transparente.

Los protocolos de comunicación empleados pueden ser:

- HTTP
- RTP
- WebRTC.

5 SEGURIDAD EN FIWARE

Este capítulo trata de estudiar las diferentes herramientas que proporciona FIWARE para mejorar la seguridad de las Smart-Solutions.

Hasta el momento, se han visto las diferentes herramientas que FIWARE pone a disposición de los desarrolladores para que lleven a cabo sus Smart-Solutions y realicen las funcionalidades que deseen. En este capítulo, se tratarán todos los aspectos de seguridad que tiene que ver con un sistema FIWARE.

Se analizarán los diferentes protocolos que se han empleado para el transporte de datos, asi como los mecanismos de seguridad proporcionados por los Generic Enablers que ofrece FIWARE, entre los que destaca Keyrock.

También se explicará cómo se relacionan el resto de Generic Enablers con cada uno de los componentes cuya función es proporcionar seguridad al sistema.

5.1 Keyrock Identity Management

Keyrock es el componente principal encargado de proveer al sistema de la seguridad necesaria para que la información de contexto sea accedida por aquellos componentes que tengan los permisos necesarios para hacerlo.

Keyrock proporciona mecanismos para gestionar la identidad, de forma que se pueda conocer en todo momento quién o qué está accediendo a la información de contexto del sistema. También proporciona una API propia con la que llevar a cabo tareas de seguridad que se detallarán mas adelante.

Para administrar Keyrock, se establece un usuario por defecto en la instalación del mismo, cuya contraseña se recomienda cambiar al momento de la instalación. Con este usuario administrador, se procede a gestionar el acceso de los diferentes componentes del sistema a los datos.

Para poder gestionar la seguridad del sistema, Keyrock ofrece una interfaz gráfica en la que se acceden a diferentes herramientas de gestión. El primer paso para empezar a gestionar la seguridad del sistema es acceder a la página de inicio de sesión y loguearse con un usuario ya existente o el usuario administrador creado al instalar esa herramienta.

El acceso a la interfaz gráfica se configura en la instalación de Keyrock. Se explica con más detalle en el anexo B.2. En nuestro caso, para acceder a la interfaz gráfica introduciremos en un navegador la dirección *http://localhost:3005* .

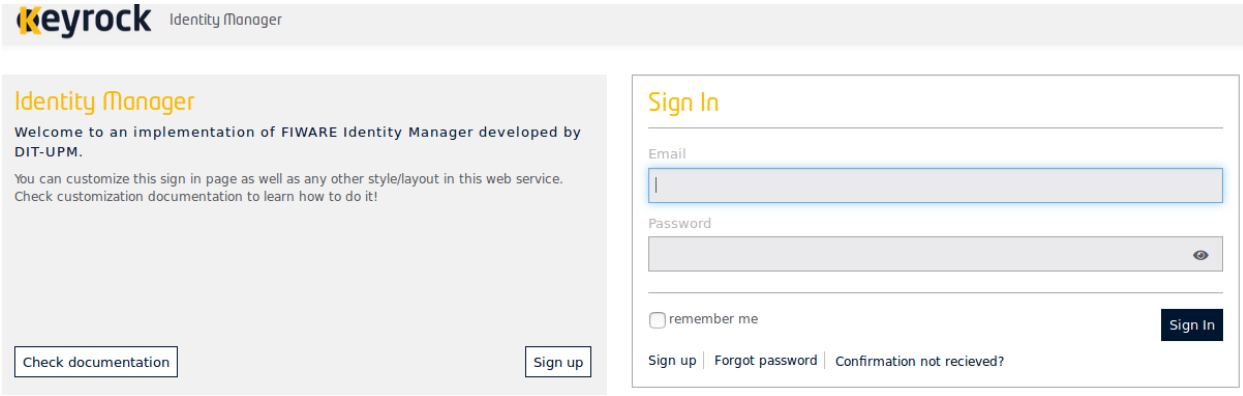


Ilustración 26: Página de inicio de sesión en Keyrock

Una vez se ha iniciado sesión con un usuario válido, se muestran las diferentes funcionalidades que dispone Keyrock, desde donde se pueden gestionar todos los aspectos de seguridad citados anteriormente.

5.1.1 Nivel Básico de Seguridad

El nivel básico de seguridad se adquiere cuando el sistema dispone de un mecanismo de autenticación capaz de tener multitud de usuarios asociados a varias aplicaciones. Cada usuario tendrá acceso a los recursos que la aplicación a la que estén registrados permita. Con este nivel podremos permitir o denegar desde Keyrock el acceso a todos o a ninguno de los recursos en función de si el usuario es válido o no.

Para hacer efectivo este nivel de seguridad es necesario tener un usuario registrado en Keyrock. Esto se puede hacer desde la página de inicio de sesión. Se entrará más en detalle en el Capítulo 6.

Una vez haya un usuario registrado en Keyrock, este debe estar asociado a una o varias aplicaciones a través de las cuales acceder a la información de contexto. Para que una aplicación pueda hacer uso de las funcionalidades de Keyrock, esta debe estar registrada. Este paso se detalla en el Capítulo 6.

Llegados a este punto, se ha registrado una aplicación y un usuario capaces de acceder a los recursos del sistema. Es decir, si un usuario está registrado en una aplicación, que a su vez está registrada en Keyrock, el usuario puede acceder a la información de contexto almacenada.

En este nivel de seguridad, Keyrock ejerce como PDP. Es decir, Keyrock es el encargado de tomar la decisión de aceptar o denegar el acceso de un usuario en función de si el usuario es válido o no [32].

5.1.1.1 Proxy PEP Wilma

Para proporcionar el nivel básico de seguridad, necesitamos de la figura del proxy PEP Wilma. Este es el componente intermedio entre la aplicación y Keyrock. Es el encargado de ejercer la decisión que toma Keyrock respecto de la petición de acceso a un recurso determinado.

En este esquema de funcionamiento, Keyrock ejerce de PDP y Wilma ejerce de PEP. Este proxy PEP es el encargado de permitir a los usuarios la capacidad de acceder a los Generic Enablers de nuestro sistema y a los servicios REST.

De hecho, el componente con el que los usuarios tendrán que interactuar mediante la API NGSI es con el proxy PEP, ya que será el componente que ejecute la decisión tomada por keyrock. Dicha respuesta la comunica el proxy PEP, informando de que se ha aceptado o denegado la solicitud.

Este componente carece de interfaz gráfica alguna, su configuración se lleva a cabo en el momento de la instalación [33].

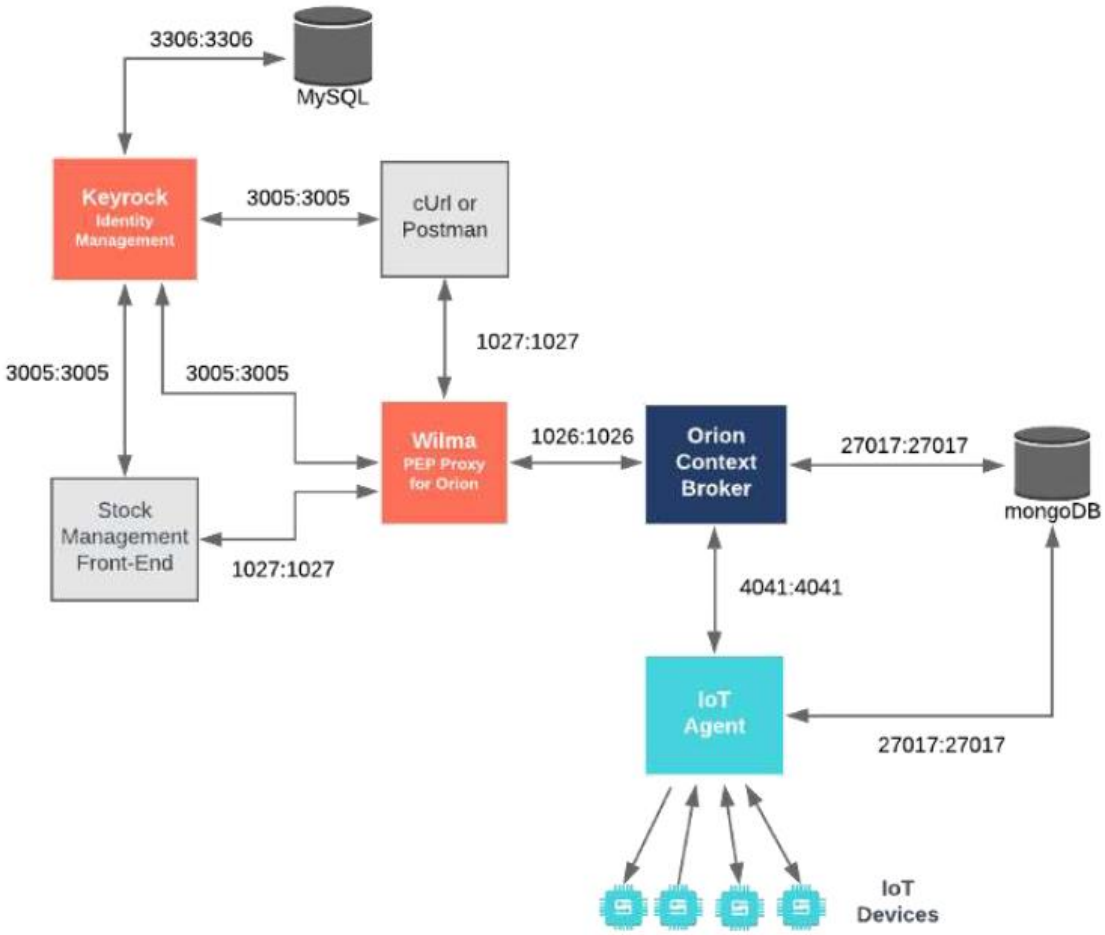


Ilustración 27: Arquitectura de un sistema con nivel de seguridad intermedio

5.1.2 Nivel Intermedio de Seguridad

El nivel intermedio de seguridad se obtiene a partir de la asignación de roles a los usuarios que componen el sistema. Una vez se hayan establecido los roles que necesite la aplicación, se establecen una serie de permisos a cada uno de estos roles. Los usuarios pueden pertenecer a una o varias organizaciones, cuya función es proporcionar permisos comunes a todos aquellos que pertenezcan a la misma organización. Por ejemplo, la organización de "administración" dispondrá de diferentes permisos que la organización de "solo acceso".

Para asignar nuevos roles se accede a la página de la aplicación y se pulsa el botón "Manage Roles". En este menú se podrán crear o eliminar los roles asociados a la aplicación. A cada uno de estos roles se les pueden asignar permisos.

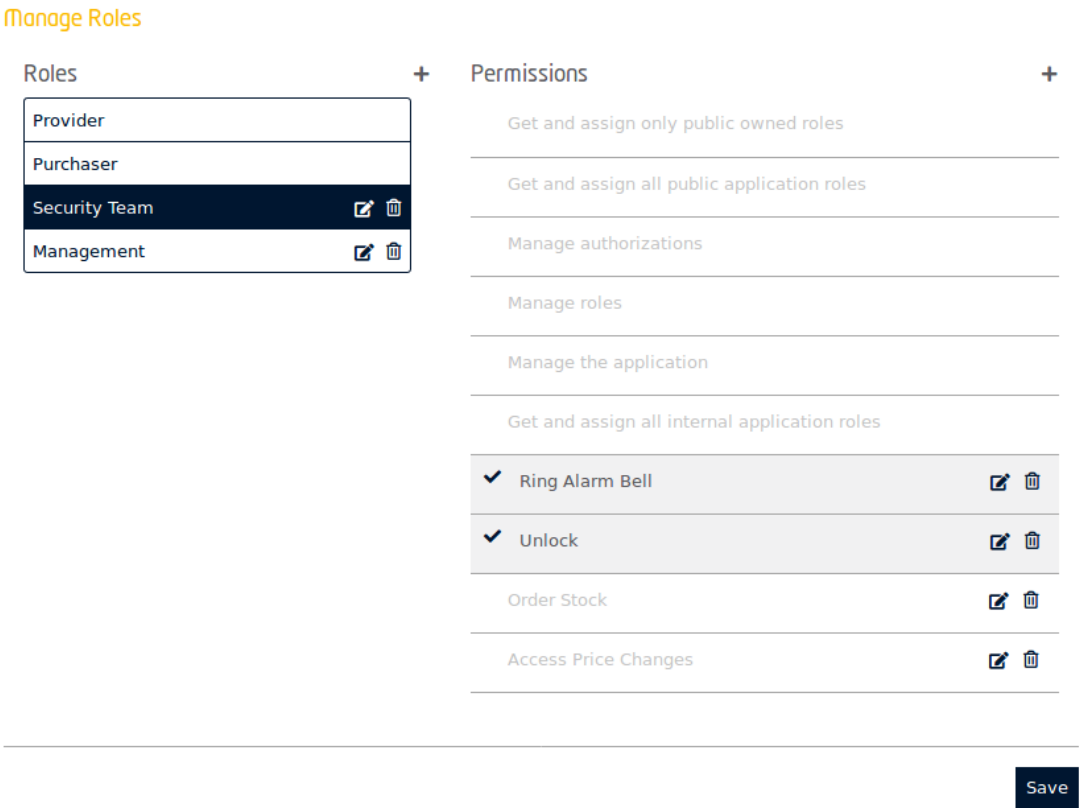


Ilustración 28: Gestión de roles y permisos en Keyrock

Para crear un nuevo permiso, basta con pulsar en el símbolo "+" situado en la parte superior izquierda de la página de "Gestión de Roles".

Para crear un nuevo permiso, se deben rellenar los siguientes campos:

- Nombre del permiso.
- Descripción del permiso
- Acción HTTP: Se decide que acción HTTP es la que se evalúa en este permiso. Suelen ser peticiones HTTP GET o HTTP POST.
- Recurso: Se indica el recurso al que se permite acceder a la aplicación. Es decir, keyrock limita las acciones que puede realizar cada usuario en función de este campo. Por ejemplo, si se desea cambiar el precio de un elemento y la URL que lleva a cabo dicha acción en la aplicación acaba en "/app/price-change", este texto es el que se pondría en el campo de recurso.

Por último, para asignar usuarios a aplicaciones, basta con entrar en el detalle de la aplicación y, en la sección "Usuarios autorizados" pulsar sobre el botón "Autorizar". Finalmente añadimos el usuario en cuestión y le asignamos un rol o roles para esta aplicación.

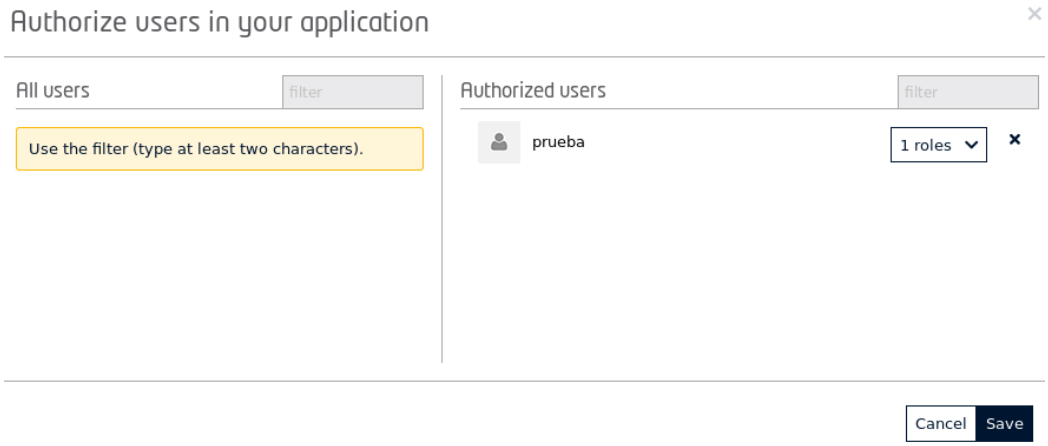


Ilustración 29: Ventana de asignación de usuarios y roles.

Con este nivel de seguridad, cada aplicación ofrece un control en el acceso a la información de contexto que maneja dicha aplicación.

5.1.3 Nivel avanzado de Seguridad

El nivel avanzado de seguridad se alcanzaría con la implementación en nuestro sistema de Authzforce, cuya función en el sistema es de ejercer de PDP. Esta herramienta proporciona una API para obtener decisiones de autorización basadas en unas políticas determinadas y en las peticiones de autorización procedentes de los PEPs.

La API proporcionada es una API REST y está compilado en *XACML*[23]. XACML es un estánar *OASIS*[24] que describe el formato de las políticas de autorización y lógica de evaluación. También se definen el formato de la solicitud o respuesta de la decisión de autorización. Todo ello recogido en un archivo en formato XML [34].

Este componente ofrece dos APIs:

- PDP API: Proporciona una API a partir de la cuál obtener decisiones de autorización definidas en XACML.
- *PAP*[25] API: Proporciona una API para la gestión de políticas XACML que serán utilizadas por el servicio de Autorización PDP.

[23] eXtensible Access Control Markup Language: Lenguaje extensible de marcas de acceso de control.
[24] Consorcio internacional orientado al desarrollo, convergencia y adopción de estándares de comercio electrónico y servicios web.
[25] Policy Administration Point: Punto de administración de Políticas

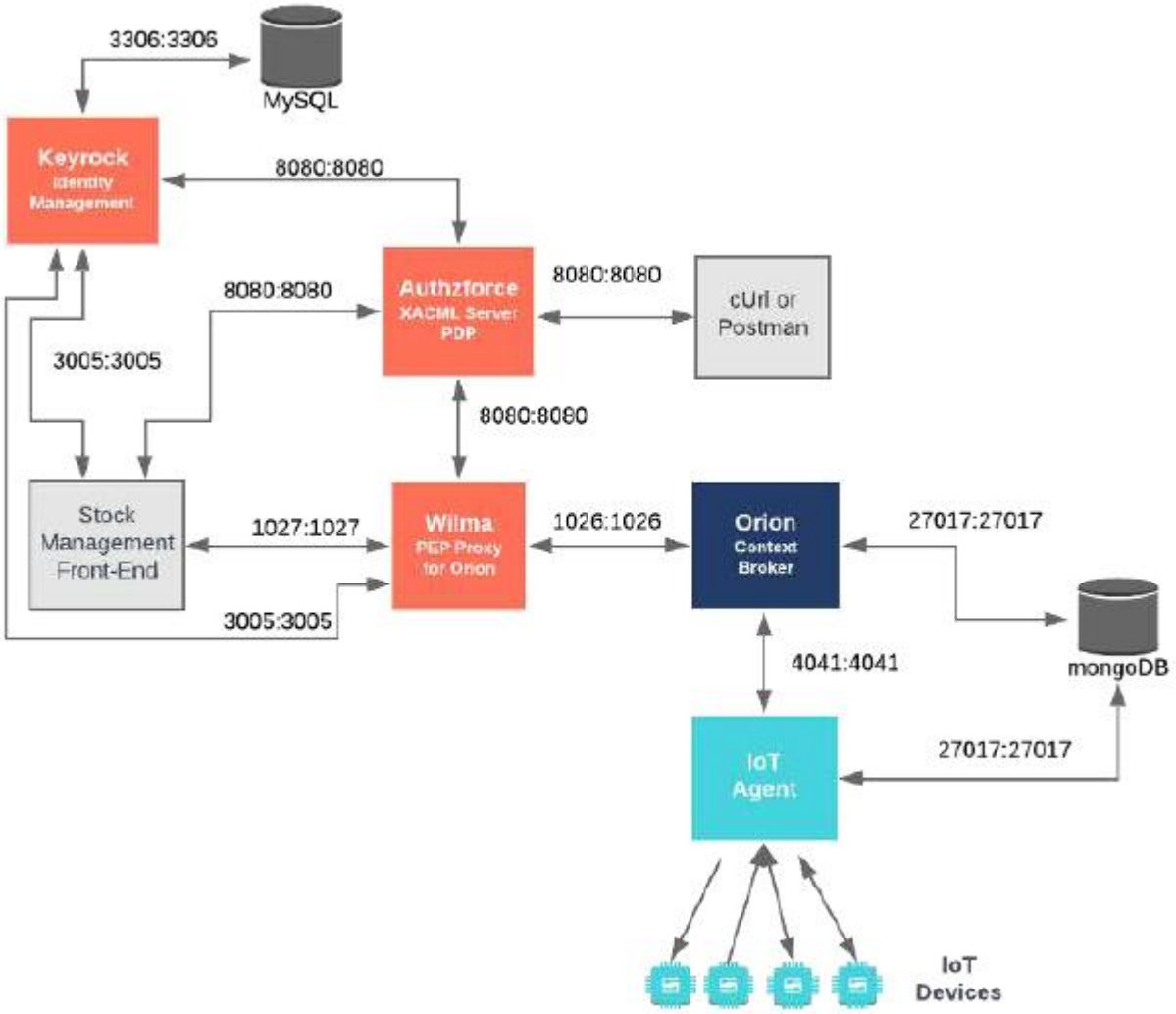


Ilustración 30: Arquitectura del sistema con nivel de seguridad avanzado.

Con esta nueva arquitectura, observamos que el PEP es el proxy PEP Wilma y el PDP es Authzforce. Keyrock deja de tener el papel de PDP en este caso, ya que delega la decisión final en Authzforce [33].

Este componente es útil si el sistema tiene una gran complejidad, ya que proporciona gran variedad de mecanismos de seguridad. Pero para la mayoría de sistemas, esta herramienta carece de utilidad al no sacarle partido y no usar la gran mayoría de utilidades que ofrece.

5.1.4 OAuth2

OAuth2 es un protocolo de autorización que surgió a partir del nacimiento de las redes sociales. Permite que los usuarios autoricen a terceros a acceder a su información sin necesidad de que estos conozcan las credenciales del usuario [35].

Las entidades involucradas en el flujo OAuth2 son:

- Propietario de recursos: Componente encargado de autorizar el acceso a los recursos protegidos. Se puede elegir la autorización a unos recursos y a otros no. En general esta figura es un humano
- Cliente: Sitio Web o aplicación que accederá a los recursos protegidos de un usuario con la autorización del mismo.

- Proveedor:
    - Servidor de autenticación: Valida usuario y credenciales y genera tokens de acceso.
    - Servidor de recursos: Recibe peticiones de acceso a los recursos protegidos autorizando el acceso si el token incluido en la petición es válido.

Las diferentes opciones para acceder a los recursos protegidos son:

- Acceso en función de credenciales: Este método es el más usado tradicionalmente. Se inicia sesión en una aplicación web y se comprueba que las credenciales son correctas viendo si coincide con los datos almacenados en el sistema. En caso positivo, el sistema otorga un token de acceso al sistema.
- Solicitud de código de autorización: La entidad cliente envía una petición con una serie de parámetros completados al propietario de los recursos. El usuario accede con sus credenciales y acepta la petición del cliente a acceder a ciertos recursos. Automáticamente se redirigirá a la url configurada incluyendo el código de autorización, que le dará acceso al recurso en cuestión. Finalmente, se intercambiará el código de autorización por un token de acceso.

Un ejemplo claro del uso de este protocolo es el registro en multitud de aplicaciones de terceros, que ofrecen la posibilidad de registro a partir de la cuenta de Facebook. El usuario accede con las credenciales de facebook y la aplicación en cuestión recoge los datos necesarios para su registro [33].

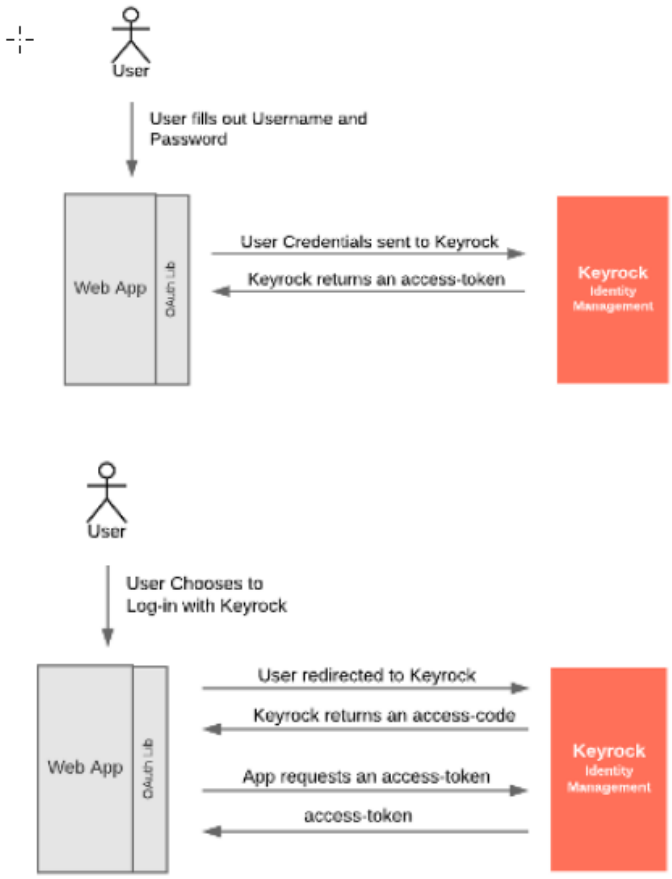


Ilustración 31: Métodos de autenticación usando el protocolo OAuth2 en Keyrock

6 SMART-SOLUTION DESARROLLADA

Este capítulo se centra en el desarrollo de una Smart-Solution basada en los Generic Enablers proporcionados por la plataforma FIWARE, haciendo especial incapié a aquellos encargados de dotar de seguridad al sistema.

La configuración de todos los componentes necesarios para el correcto funcionamiento del sistema está detallado en el Anexo B.

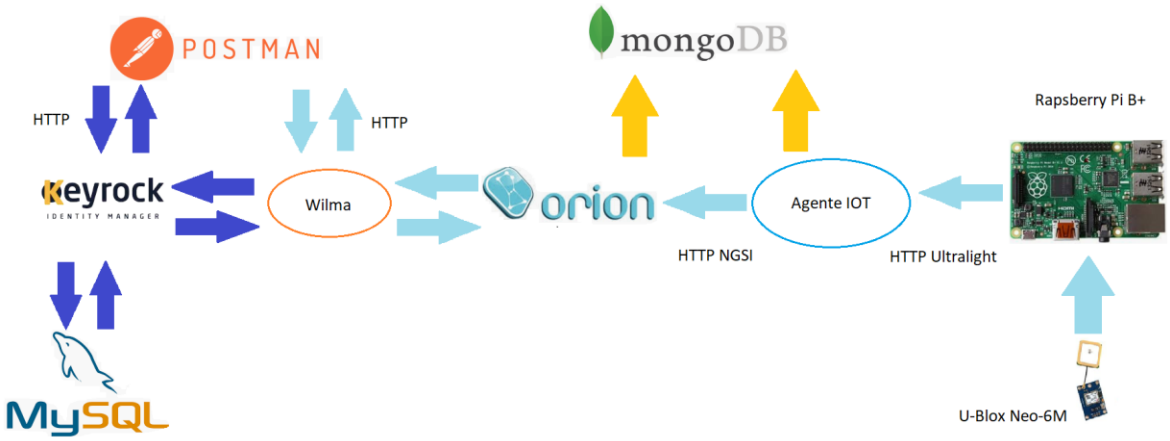


Ilustración 32: Arquitectura del sistema

6.1 Recogida de datos del agente IOT

En este apartado se procede a detallar las acciones realizadas para poder recoger los datos que envía la Raspberry Pi en el formato correcto.

6.1.1 Creación del servicio MOVE

En primer lugar, es necesario crear un servicio que gestione los datos que se envíen a cierto recurso. Para ello, se le indica al agente IOT por el puerto norte la siguiente información:

```
curl -X POST \
  http://localhost:4041/iot/services \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 81279e80-b637-419d-819b-01aed435a51d' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://localhost:1026",
     "entity_type": "Move",
     "resource":    "/iot/d"
   }
 ]
}'
```

Ilustración 33: Mensaje POST enviado al agente IOT.

En esta petición POST se indican las siguientes cabeceras:

- Fiware-service: Establece el nombre del grupo de servicios creado, en este caso, el grupo de servicios se llama “TFG”.
- Fiware-servicepath: Se trata de la ruta que se debe seguir para acceder a la información de este grupo de servicios. Está pensado para construir una jerarquía de dispositivos en función, por ejemplo, de su funcionalidad.
- La entidad services formado por los campos:
    - API Key: Clave conocida que se debe proporcionar siempre que se quiera escuchar o comunicarse con la entidad de servicio en cuestión. Esta cabecera se debe indicar en la petición post que haga el programa gps.py para poder comunicarse con el agente IOT.
    - Entity-type: Tipo de la entidad en la que se almacenarán los datos que los dispositivos IOT envíen al agente IOT.
    - Cbroker: Indica la dirección y puerto del Context Broker con el que se debe mantener la comunicación.
    - Resource: Indica al recurso que se deben enviar los mensajes que contengan información de contexto recogida por los sensores.

Gracias a este paso, podemos los diferentes datos que nos lleguen al recurso determinado a un tipo de entidad concreto. Lo ideal es que cada dispositivo IOT envie su información a un recurso concreto.

El agente IOT manda un mensaje de respuesta indicando el estado **201 Created**.

6.1.2 Asociar el sensor a un servicio

Se procede a incluir el dispositivo encargado de enviar la información de contexto, en este caso la Raspsberry, que enviará esta información según el formato que se configure en este paso.

La petición POST que le realizaremos al agente IOT es la siguiente:

```
curl -X POST \
  http://localhost:4041/iot/devices \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 22d81d36-09ec-4912-b184-a1e7b820c427' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
  {
    "device_id":   "Move01",
    "entity_name": "urn:ngsi-ld:Move01",
    "entity_type": "Move",
    "timezone":    "Europe/Madrid",
    "attributes": [
      { "object_id": "lat", "name":"latitud", "type":"Float"},
      { "object_id": "lon", "name":"longitud", "type":"Float"},
      { "object_id": "alt", "name":"altitud", "type":"Float"},
      { "object_id": "vel", "name":"velocidad", "type":"Float"},
      { "object_id": "dir", "name":"direccion", "type":"Float"},
      { "object_id": "sub", "name":"subida", "type":"Float"},
      { "object_id": "fec", "name":"fecha", "type":"String"}
    ]
  }
 ]
}
```

Ilustración 34: Creación de la entidad encargada de almacenar la información de contexto.

En esta petición se observan las siguientes cabeceras:

- Fiware-service: Indica el grupo de servicio al que pertenece el disposirtivo.
- Fiware-servicepath: Se trata de la ruta que se debe seguir para acceder a la información de este grupo de servicios.
- Campos referidos al dispositivo en cuestión:
    - Device_id: Identifica unívocamente al disposirivo dentro del sistema.
    - Entity_name: Nombre único de la entidad que se está creadno para gestionar la información del sensor GPS.
    - Entity_type: El tipo de la entidad.
    - Timezone: La zona horaria.
    - Attrinbutes:Conforman cada uno de los atributos que dispone la entidad. Como se puede observar, se establecen diferentes parámetros para cada atributos.
        - Object_id: Este será la clave del conjunto clave valor que enviarán los disposirivos IOT. El agente IOT será el encargado de relacionar estos ids con el nombre del atributo.
        - Name: Nombre del atributo.
        - Type: Tipo del atributo.

El agente IOT manda un mensaje de respuesta indicando el estado **201 Created**.

6.2 Captación de datos de la Rapsberry Pi

Para recoger datos de interés, se ha empleado un sensor GPS U-Blox NEO-6M, el cuál se ha conectado a una Raspberry Pi Model B+ mediante el protocolo UART.

Para la recogida y posterior envío de datos por parte de la Raspsberry se ha realizado el programa python gps.py.

```
import os
from gps import *
from time import *
import time
import threading
import requests

gpsd = None #seting the global variable

os.system('clear') #clear the terminal (optional)

class GpsPoller(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        global gpsd #bring it in scope
        gpsd = gps(mode=WATCH_ENABLE) #starting the stream of info
        self.current_value = None
        self.running = True #setting the thread running to true

    def run(self):
        global gpsd
        while gpsp.running:
            gpsd.next() #this will continue to loop and grab EACH set of gpsd info to clear the buffer

if __name__ == '__main__':
    gpsp = GpsPoller() # create the thread
    try:
        gpsp.start() # start it up
        while True:
            #It may take a second or two to get good data
            #print gpsd.fix.latitude,', ',gpsd.fix.longitude,'  Time: ',gpsd.utc

            os.system('clear')

            print (' GPS reading')
            print ('----------------------------------------')
            print ('latitude    ' , gpsd.fix.latitude)
            print ('longitude   ' , gpsd.fix.longitude)
            print ('time utc    ' , gpsd.utc,' + ', gpsd.fix.time)
            print ('altitude (m)' , gpsd.fix.altitude)
            print ('eps         ' , gpsd.fix.eps)
            print ('epx         ' , gpsd.fix.epx)
            print ('epv         ' , gpsd.fix.epv)
            print ('ept         ' , gpsd.fix.ept)
            print ('speed (m/s) ' , gpsd.fix.speed)
            print ('climb       ' , gpsd.fix.climb)
            print ('track       ' , gpsd.fix.track)
            print ('mode        ' , gpsd.fix.mode)
            print('-')
            datos = "lat|"+ str(gpsd.fix.latitude) + "|lon|"+ str(gpsd.fix.longitude) + "|alt|"+ str(gpsd.fix.altitude) +
            "|vel|"+ str(gpsd.fix.speed) + "|dir|"+ str(gpsd.fix.track) + "|sub|"+ str(gpsd.fix.climb) + "|fec|"+ str(gpsd.fix.time)
            headers = {'content-type' : 'text/plain'}
            req = requests.post('http://192.168.1.206:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=Move01', data=datos, headers=headers)
            print ("Enviando los datos obtenidos del sensor GPS: " + datos)
            time.sleep(15) #set to whatever

    except (KeyboardInterrupt, SystemExit): #when you press ctrl+c
        print ("\nKilling Thread...")
        gpsp.running = False
        gpsp.join() # wait for the thread to finish what it's doing
    print ("Done.\nExiting.")
```

Ilustración 35: gps.py

Para verificar el correcto funcionamiento del programa python, asi como la recogida de datos, se muestra por pantalla los datos que recoge el sensor en tiempo real. El resultado de la ejecución debe ser el siguiente:

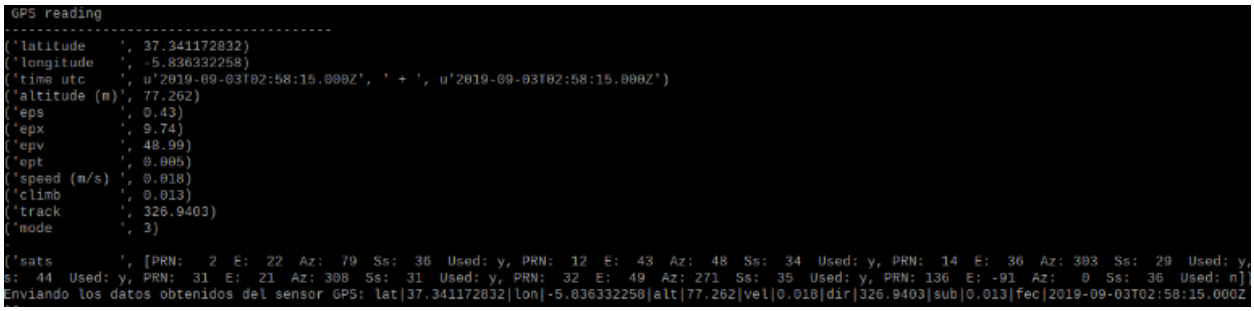


Ilustración 36: Resultado de la ejecución del programa gps.py

Cabe destacar que el envío de mensajes HTTP desde la Raspberry Pi hasta el agente IOT se hace a través de WiFi. EL protocolo que se ha elegido para enviar el cuerpo de los mensajes es el protocolo Ultralight, el cuál se ha detallado anteriormente.

6.3 Acceso a la información de contexto

En este apartado se explicará cómo conseguir un acceso seguro a la información de contexto, gracias a los mecanismos de seguridad estudiados en el Capítulo 5.

6.3.1 Acceso no seguro

En primer lugar, se procede a la comprobación de que Orion Context Broker está recibiendo la información correctamente procedente de la Raspberry Pi. Para ello, se hará una petición directamente al Context Broker, el cuál no proporciona ningún mecanismo de seguridad. Se realiza entonces la siguiente petición:

```
curl -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: e4cd9775-d32d-46a8-b523-438dc35234e0' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

Ilustración 37: Petición HTTP GET a Orion para consultar los valores del dispositivo Move01

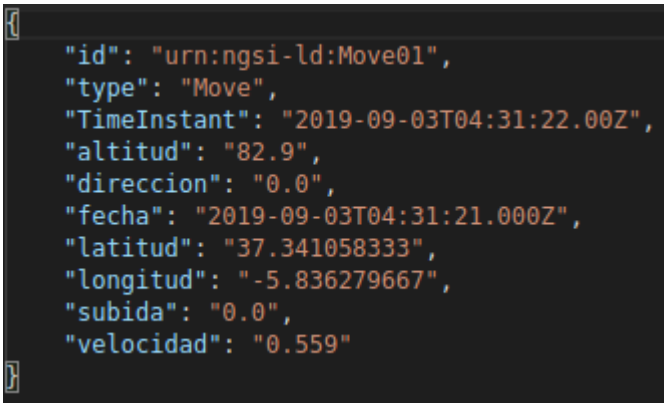
Esta petición cuenta con las siguientes cabeceras:

- Fiware-service: TFG, grupo de servicios creado en el apartado 6.1.1
- Fiware-servicepath: Path que indica la ruta de la información de contexto del dispositivo

También se pueden observar otros datos de interés en la URL:

- Urn:ngsi-ld:Move01: Se trata del nombre de la entidad que se creo en el apartado 6.1.2 a la que se quiere acceder para consultar su informaciónde contexto
- Options=keyValues: Indica que solo queremos la información de las parejas clave/valor.

La respuesta a dicha petición es la siguiente:



```
{
    "id": "urn:ngsi-ld:Move01",
    "type": "Move",
    "TimeInstant": "2019-09-03T04:31:22.00Z",
    "altitud": "82.9",
    "direccion": "0.0",
    "fecha": "2019-09-03T04:31:21.000Z",
    "latitud": "37.341058333",
    "longitud": "-5.836279667",
    "subida": "0.0",
    "velocidad": "0.559"
}
```


Ilustración 38: Parejas clave/valor del dispositivo Move01

Hasta este momento, el sistema funciona sin mecanismos de seguridad, ya que no se han hecho uso de ningún herramienta de las estudiadas en apartados anteriores.

6.3.2 Acceso seguro

En este apartado se proceden a detallar los pasos necesarios que se deben dar para dotar de seguridad al sistema.

En nuestro sistema no disponemos de aplicación web con interfaz gráfica. Sin embargo, para poder realizar peticiones seguras al Context Broker ha sido necesario el registro de una aplicación en Keyrock en la que iniciar sesión para hacer uso del nivel básico de seguridad. En este proyecto se ha hecho uso de esta aplicación mediante peticiones HTTP desde Postman, es decir, Postman será la interfaz gráfica de nuestra "aplicación". A partir de ahora esta aplicación será llamada "aplicación ficticia".

En nuestro caso, el sistema tendrá un nivel de seguridad básico, en el que tendremos un usuario que accederá a la información de contexto a través de esta aplicación ficticia.

6.3.2.1 Registro de usuario en Keyrock

Se procede a detallar los pasos necesarios que hay que seguir para dotar de un nivel de seguridad básico a nuestro sistema.

En primer lugar, se va a crear un usuario en Keyrock, el cuál será el encargado de acceder a la información de contexto de forma segura [30].

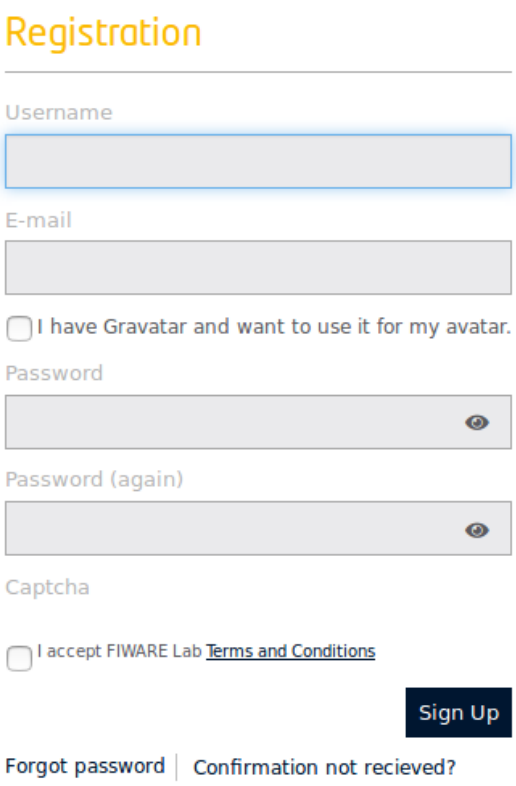


Ilustración 39: Registro de usuario en Keyrock

Para ello, se accede a la página de inicio de sesión *http://localhost:3005* y se registra por primera vez, completando los campos:

- Nombre de usuario
- E-mail
- Password (2 veces para confirmar)
- Aceptar los términos y condiciones de FIWARE Lab.

Una vez se cree el usuario, el administrador debe activarlo desde la página de gestión de usuarios. Estos nuevos usuarios se almacenan en la base de datos mySQL[31].

6.3.2.2 Registro de aplicación en Keyrock

Una vez se dispone del usuario para acceder a la herramienta Keyrock, se inicia sesión.

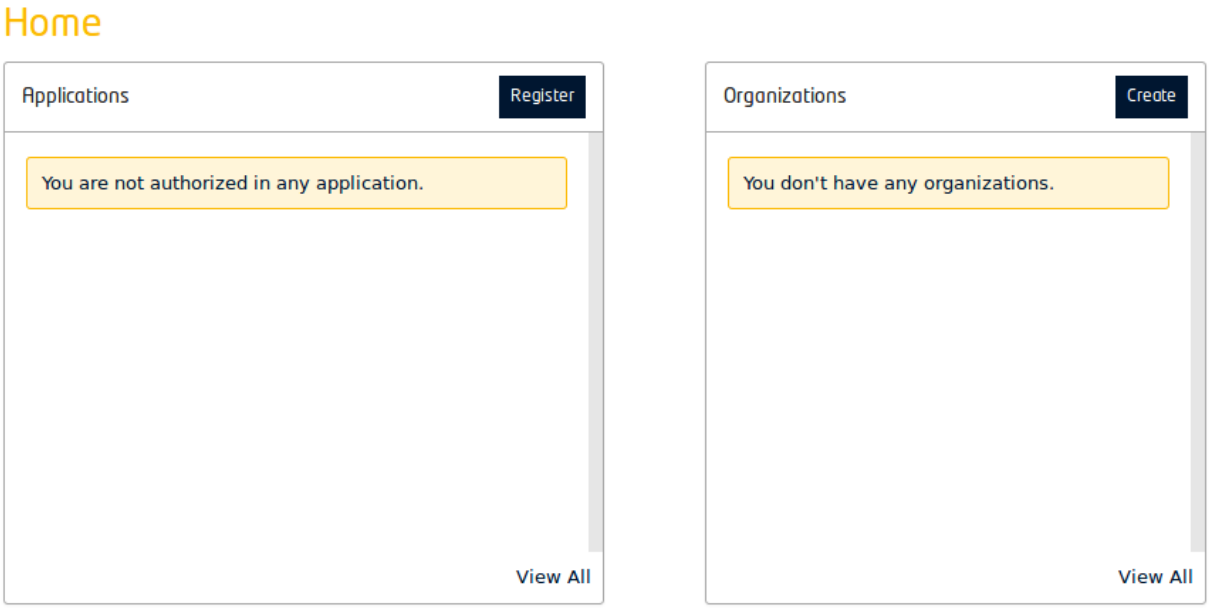


Ilustración 40: Página de bienvenida de Keyrock

En segundo lugar, se ofrece un mecanismo de registro de aplicaciones que harán uso de este Generic Enabler. Para registrar una aplicación es necesario completar los siguientes campos:

- Nombre de la aplicación.
- Descripción de la aplicación.
- URL en la que se encuentra alojada la aplicación.
- Callback URL: URL a la que Keyrock redirigirá el flujo una vez se haya completado el proceso de autenticación.
- Sign-out Callback URL: URL a la que Keyrock redirigirá el flujo una vez se haya completado el cierre de la sesión del usuario, tanto de la aplicación como de Keyrock.
- Grant Type: Se establecen los métodos que se podrán emplear en la aplicación para verificar la identidad del usuario, los cuales los proporciona el protocolo de autorización OAuth2, detallados posteriormente.
- Provider: Establece el usuario propietario de la aplicación o la organización encargada.

En este proyecto, no es necesaria la configuración de la aplicación, ya que no se dispone de aplicación alguna. Se carece de aplicación web que haga uso de los datos recogidos. También se prescinde de dashboards, ya que la mayoría no son compatibles con las funcionalidades que ofrece Keyrock. No obstante, se puede comprobar la seguridad del sistema simplemente mediante la autenticación de los usuarios, sin importar a qué aplicación pertenezcan.

Una vez registrada la aplicación, se entra al detalle de dicha aplicación. Ahora, es necesario copiar los campos “Client ID”y “Client ID Secret” concatenarlos en una línea y separado por el caracter “:”, con ClientIdSecret a continuación de ClientID.

Posteriormente, se busca un codificador base 64 y se anota el resultado de la codificación.

6.3.2.3 Acceso autorizado a la información de contexto

Una vez hemos tenemos todos los requisitos, solicitamos información de contexto al proxy PEP, ya que es el componente con el que debemos comunicarnos para acceder a los datos almacenados en el context broker de forma segura.

En primer lugar, se inicia sesión para que Keyrock nos conceda un token de acceso para acceder a los recursos de nuestro sistema.

```
curl -X POST \
  http://localhost:3005/oauth2/token \
  -H 'Accept: application/json' \
  -H 'Authorization: Basic dHV0b3JpYWwtZGNrci1zaXRlLTAwMDAteHByZXNzd2ViYXBwOnR1dG9yaWFsLWRja3Itc2l0ZS0wM
      DAwLWNsaWVudHNlY3JldA==' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H 'Postman-Token: 58c7dfca-9bef-4189-b2ab-48578842d727' \
  -H 'cache-control: no-cache'
```

Ilustración 41: Petición de un token de acceso a Keyrock

En esta petición se pueden observar las siguientes cabeceras:

- Authorization: Basic dhV….: Esta cabecera indica el nivel de seguridad del sistema junto al client id y client id secret codificado en base 64.
- El cuerpo del mensaje es el siguiente:

En el cuerpo del mensaje simplemente se indican las credenciales y el tipo de autenticación que van a llevar a cabo.

```
username=javier@test.com&password=test&grant_type=password
```

Ilustración 42: Cuerpo de la petición de token de acceso a la aplicación.

La respuesta, en caso de que el usuario sea válido, es la siguiente:

```
{
    "access_token": "bec6bbc45655af2cc340b911b9e391f92b9bf7a4",
    "token_type": "Bearer",
    "expires_in": 3599,
    "refresh_token": "8a2172bb8ad034782804aed9c3f9e3a707b8b425",
    "scope": [
        "bearer"
    ]
}
```

Ilustración 43: Respuesta correcta da la petición de un token de acceso

Una vez obtenido el token de acceso que permite consultar la información de contexto almacenada en el Context Broker, se procede a realizar una petición de consulta de los datos recogidos por el sensor GPS al proxy PEP. Es importante diferenciar las peticiones, ya que la comunicación con el PEP Proxy requiere de un token de acceso siempre, mientras que si se establece la comunicación con Orion Context Broker, este no solicita ningún tipo de autenticación.

```
curl -X GET \
  'http://localhost:1027/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: ec37de1d-5079-4e50-b686-2acccd6d49e7' \
  -H 'X-Auth-Token: bec6bbc45655af2cc340b911b9e391f92b9bf7a4' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

Ilustración 44: Petición GET al proxy PEP para obtener los datos del sensor GPS con token válido

Como se puede observar, se requiere de una cabecera de nombre "X-Auth-Token" en la que va una cadena aleatoria generada previamente por Keyrock y que se ha recuperado del mensaje de respuesta al inicio de sesión. También es necesaro incluir el nombre del grupo de servicio (TFG) y el fiware-servicepath (/).

La respuesta a dicha petición es:


```
{
    "id": "urn:ngsi-ld:Move01",
    "type": "Move",
    "TimeInstant": "2019-09-03T05:17:54.00Z",
    "altitud": "60.7",
    "direccion": "0.0",
    "fecha": "2019-09-03T05:17:54.000Z",
    "latitud": "37.341122333",
    "longitud": "-5.8363185",
    "subida": "0.0",
    "velocidad": "0.01"
}
```


Ilustración 45: Respuesta de la petición GET que pedía la información de contexto de MOVE.

6.3.2.4 Acceso no autorizado a la información de contexto

Para ilustrar un ejemplo de acceso no deseado, se procede a hacer una petición a los mismos recursos que en el apartado anterior en el que el token de acceso es inválido.

```
curl -X GET \
  'http://localhost:1027/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: ba8f2531-36aa-4782-9e39-6e9fb80a0097' \
  -H 'X-Auth-Token: ************************************' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

Ilustración 46: Petición GET al proxy PEP para obtener los datos del sensor GPS con token no válido

El mensaje de respuesta será el siguiente:

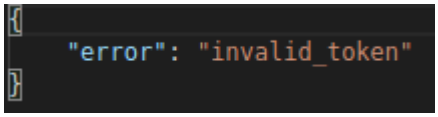


Ilustración 47: Mensaje de respuesta
al enviar un token erróneo

Por tanto, se ha demostrado que el sistema tiene un nivel de seguridad básico, ya que es capaz de permitir o denegar el envío de información de contexto en base a si un usuario es válido o no.

7 CONCLUSIONES Y LÍNEAS FUTURAS

7.1 Conclusiones

La realización de este proyecto ha hecho que descubra infinidad de herramientas y plataformas que no conocía hasta el momento. El IoT es un concepto que desconocía y, gracias a la plataforma FIWARE, he descubierto multitud de herramientas que facilitan en gran medida el desarrollo de smart-solutions que hagan uso del IoT.

FIWARE proporciona una documentación muy extensa, lo cuál permite al desarrollador aprender a utilizar todos y cada uno de los componentes del sistema de forma guiada. Ofrece información detallada de todos los Generic Enablers disponibles y la funcionalidad que tiene cada uno de ellos. Además, pone a disposición del desarrollador aplicaciones que ya se han desarrollado previamente con este conjunto de herramientas que proporciona la plataforma FIWARE.

Al aprender a utilizar las herramientas de FIWARE, he descubierto tecnologías muy interesantes como Docker, la cuál conocía al haber leído algún artículo pero nunca lo había usado. A partir de este proyecto, he investigado y aprendido a usar esta tecnología, junto con Docker-compose.

También ha sido necesario el aprendizaje de lenguajes de programación como Python, ya que lo desconocía hasta la actualidad. Este lenguaje de programación es el más usado para el desarrollo de aplicaciones de Rapsberry Pi que accedan a los datos que se le envía a través de los pines de entrada.

En definitiva, la realización de este proyecto me ha descubierto un mundo que antes no había explorado, como es el IoT. A partir del estudio e investigación de las múltiples soluciones que permiten interactuar con estos dispositivos IOT, he descubierto que existe una gran variedad de oportunidades que aún están por desarrollar, y, en un futuro, me gustaría dedicarme a la realización de alguna de estas soluciones.

7.2 Líneas futuras

En este apartado, se procede a detallar algunas ideas que pretenden ampliar las funcionalidades llevadas a cabo en el presente proyecto:

- **Aplicación Web:** La incorporación de una aplicación web al sistema desarrollado en el presente proyecto, otorgaría multitud de ventajas respecto al estado actual del mismo:
    - Permitiría poder establecer niveles de seguridad intermedios y avanzados, al disponer de recursos de nivel de aplicación a los que asignar permisos.
    - Establecería una comunicación más intuitiva con el sensor GPS que se usa en el presente proyecto.
    - Ofrecería una interfaz de usuario que proporciona un nivel más alto de abstracción en el que no es necesario que el usuario conozca el acceso a la API NGSI ni el formato de las peticiones HTTP.
- **Integración con Wirecloud:** Una de las herramientas que he descubierto durante la realización del proyecto es Wirecloud, el cuál se ha explicado en el apartado 4.2.3.1. Ofrece multitud de herramientas que encajarían de forma perfecta con el sistema actual. Proporcionaría una visualización de los datos más intuitivo que los mensajes JSON que se reciben en la actualidad.
- **Desarrollo de una app para móviles.** También sería interesante la integración de una app para móviles que indique la información GPS que recoge el sensor del propio móvil. Con esta información se pueden realizar multitud de aplicaciones, como por ejemplo una aplicación que ubique permanentemente a un menor de edad en un mapa para estar localizado en todo momento.
- **Integración del sistema con Authzforce.** Como hemos explicado en este proyecto, Authzforce proporciona multitud de mecanismos de seguridad. A medida que vayamos haciendo mas complejo un sistema, va creciendo la necesidad de utilizar Authzforce para proveeder de seguridad al sistema completo.

ANEXO A: CONFIGURACIÓN DE RASPBERRY PI Y SENSOR GPS

En este capítulo se procede a explicar de forma detallada los pasos que han sido necesarios para el correcto funcionamiento de la Raspberry Pi B+ junto con el sensor GPS U-Blox 6M.

Para poder empezar a configurar la Raspberry Pi, es necesaria la instalación del Sistema Operativo Raspbian. Este sistema Operativo se debe instalar en una tarjeta de memoria microSD de al menos 4 GB. Raspberry Pi utiliza como dispositivo de arranque la tarjeta SD, pero, una vez se ejecute por primera vez el sistema, se puede configurar la Raspberry para arrancar a partir de una memoria USB.

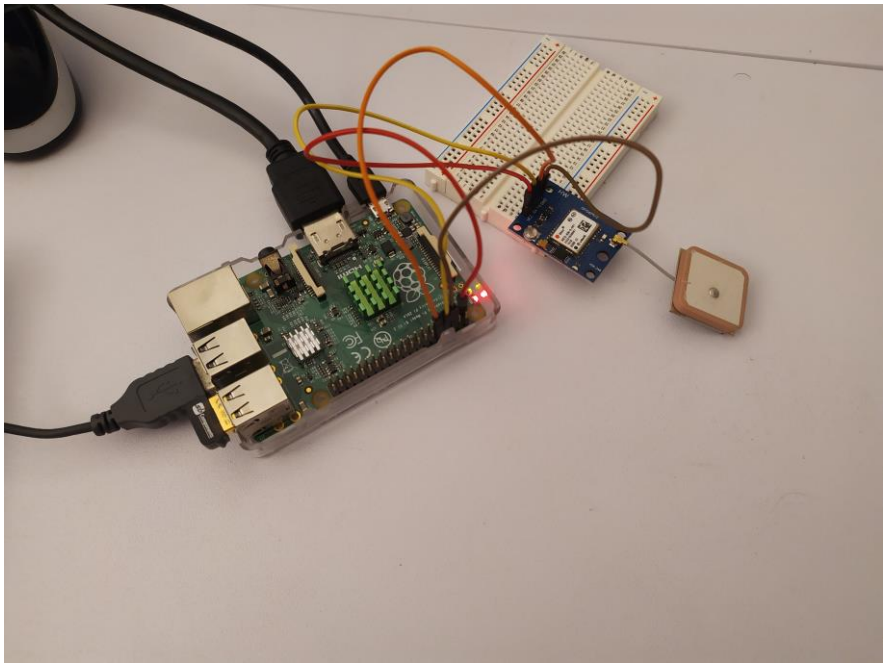


Ilustración 48: Raspberry Pi Model B+ conectado a sensor U-Blox NEO-6M

Para llevar a cabo las conexiones y configuración de la Raspberry Pi, se han seguido los pasos que detalla Luis Martínez Ruiz en su Trabajo de Fin de Grado, junto con el apoyo de alguna referencia extra.

El requisito indispensable para que nuestro sistema funcione, es la de estar conectado a internet. En nuestro caso, se ha utilizado un disposirivo Wireless USB Network para conectarnos a internet vía Wi-Fi.

A.1 Sensor U-Blox NEO-6M

Se procede a indicar las conexiones realizadas entre el sensor GPS y al Rapsberry Pi, asi como la configuración necesaria en la Raspberry Pi para poder establecer la comunicación.

Conexión

Para llevar a cabo la conexión de ambos dispositivos, se necesitan 4 cables macho/hembra y una protoboard en la que establecer las conexiones. Las conexiones se muestran en la siguiente tabla:

Sensor GPS	Raspberry Pi	Color del cable
VCC	PIN 1 (3.3V)	Rojo
GND	PIN 6 (GND)	Marrón
RX	PIN 8 (TXD0)	Amarillo
TX	PIN 10 (RXD0)	Naranja

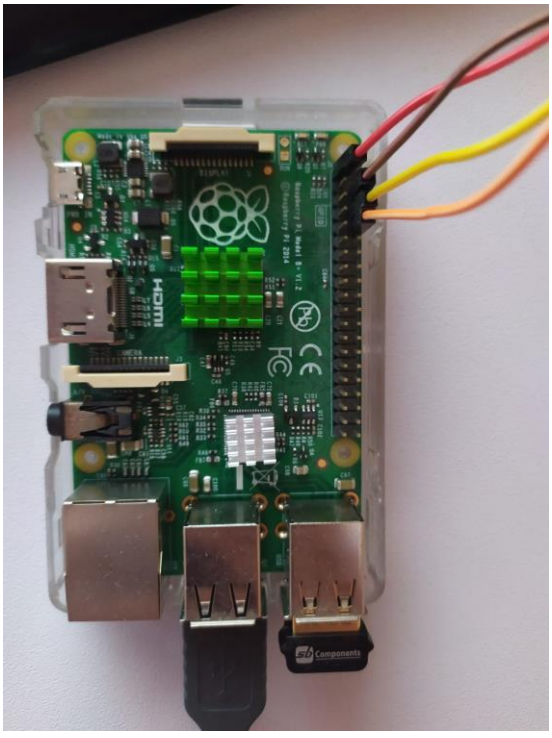


Ilustración 49: Conexiones de Raspberry Pi Model B+

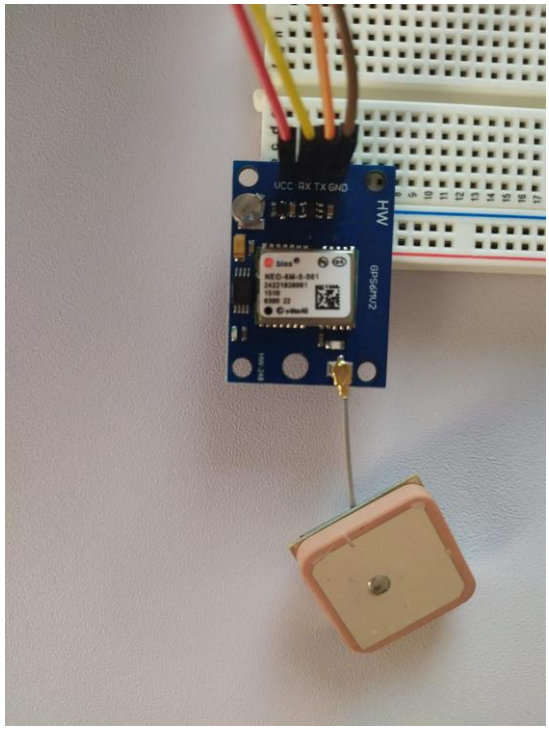


Ilustración 50: Conexiones del sensor GPS

Configuración de la Raspberry Pi Model B+

Se procede a detallar los pasos que se han seguido para configurar la Raspberry Pi y poder adquirir los datos que recoge el sensor GPS

1. Abrir al archivo */boot/cmdline.txt* como root y copiar el siguiente contenido:

```
dwc_otg.lpm_enable=0 console=tty1 root=PARTUUID=32c518e3-02 rootfstype=ext4
elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles
```

2. Abrir el archivo */boot/config.txt* y añadir la siguiente línea, la cuál habilita la comunicación UART.

```
enable_uart=1
```

3. Instalar las dependencias necesarias para obtener los datos del sensor GPS correctamente

```
$ sudo apt-get install gpsd gpsd-clients python-gps
```

4. Modificar el archivo */etc/default/gpsd* para que tenga el siguiente contenido:

```
# Default settings for the gpsd init script and the hotplug wrapper.

# Start the gpsd daemon automatically at boot time
START_DAEMON="true"

# Use USB hotplugging to add new USB devices automatically to the daemon
USBAUTO="true"

# Devices gpsd should collect to at boot time.
# They need to be read/writeable, either by user gpsd or the group dialout.
DEVICES=""

# Other options you want to pass to gpsd
GPSD_OPTIONS="/dev/ttyAMA0"
GPSD_SOCKET="/var/run/gpsd.sock"
```

5. Ejecutar el siguiente comando para reiniciar la Raspberry Pi y obtenga la configuración deseada.

```
$ sudo reboot
```

6. Finalmente, se ejecuta el demonio gpsd, el cuál se encarga de extraer los datos de interés del sensor GPS por el puerto UART. Esto permite que el programa gps.py puede hacer uso de estos datos y los envíe en el formato correcto al resto de componentes del sistema.

```
$ cgps -s
```

Este comando nos muestra por pantalla los datos que recoge en tiempo real del GPS.

Para comprobar que todo funciona según lo esperado, ejecutamos el programa gps.py, el cuál se detalla en la Ilustración 34.

```
$ python gps.py
```

El resultado esperado de la ejecución de este programa es el siguiente:

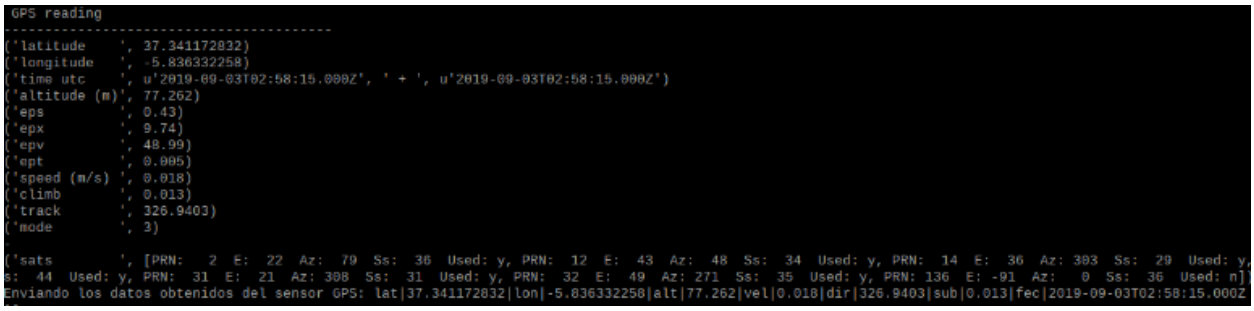


Ilustración 51: Resultado de la ejecución del programa gps.py

Se observa en la última línea de la Ilustración 48, el mensaje que se envía al agente IOT. Como se indicó en el apartado 6.1, el formato del mensaje es Ultralight.

ANEXO B: INSTALACIÓN DE LOS COMPONENTES DEL SISTEMA

En este apartado se exponen los pasos que se han seguido para instalar todos los componentes que se han utilizado en el sistema. Todos estos componentes se han instalado en contenedores gestionados por la tecnología Docker.

B.1 Instalación de Docker y Docker Compose

Docker

Los requisitos para la instalación de Docker en el Sistema Operativo Ubuntu son:

- Disponer de las versión 16.04 LTS o más reciente.
- Tener un procesador con una arquitectura de 64 bits.

La instalación se llevará a cabo a partir del repositorio.

1. Actualizar sistema de gestión de paquetes apt

```
$ sudo apt-get update
```

2. Instalar los paquetes necesarios que permiten a apt acceder a un repositorio a través de https.

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

3. Añadir la clave GPG oficial de Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

4. Instalar el repositorio estable desde el que descargar la última versión de Docker:

```
$ sudo apt-get install docker-ce
```

5. Finalmente, se installa el recurso Docker:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Un paso adicional es comprobar que el recurso se ha instalado correctamente. Esto se puede hacer mediante el comando:

```
$ docker --version
```

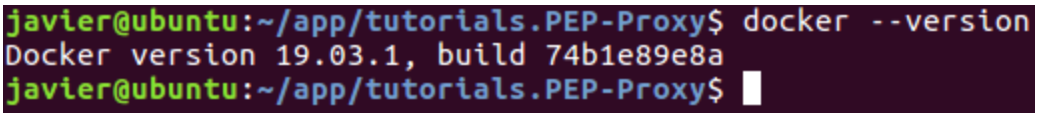


Ilustración 52: Resultado de la ejecución del comando docker --version

Docker-compose

El siguiente paso es instalar el software Docker-componse, el cuál se encarga de ejecutar varios contenedores a la vez. También permite la conexión entre estos contenedores para que funcionen como un único sistema.

1. Descargar la versión actual estable de Docker-compose desde la plataforma github. Para ello ejecutaremos el siguiente comando:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Una vez obtenido el recurso, le otorgamos permisos de ejecución:

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Al igual que en la instalación de Docker, se ejecuta el siguiente comando para comprobar que se ha realizado correctamente la instalación:

```
$ docker-compose --version
```

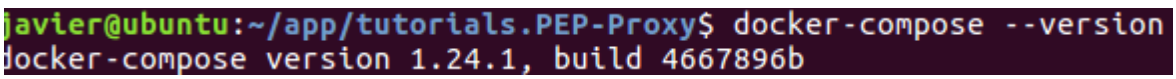


Ilustración 53: Resultado de la ejecución del comando docker-compose --version

B.2 Creación de los componentes del sistema

En este apartado se va a detallar cómo se han creado cada uno de los componentes del sistema y las herramientas utilizadas para ello.

Cabe destacar que todos los componentes están definidos a partir de imágenes en un único archivo llamado orion.yml, al cuál se accede desde Docker-compose. Para facilitar el uso de de esta herramienta y evitar la ejecución multitud de comandos, se ha adaptado un script de servicio que proporciona FIWARE, el cual se encarga de llevar a cabo todas las tareas necesarias para poner en marcha nuestro sistema.

Script de servicio

Se pueden usar los siguientes comandos:

- ./services help: Muestra los comandos disponibles.
- ./services orion: Se encarga de levantar todos los contenedores situados en el archivo orion.yml
- ./services stop: Se encarga de para y eliminar los contenedores que están ejecutandose actualmente.
- ./services create: Se encarga de hacer pull[26] de las necesarias para poder llevar a cabo el posterior arranque el sistema completo. Estas imágenes están almacenadas en la variable "image" dentro de cada contenedor en el archivo orion.yml.

```
command="$1"
case "${command}" in
    "help")
        echo "usage: services [create|orion|iot-agent]"
        ;;
    "orion")
        stoppingContainers
        echo -e "Starting six containers \033[1;34mOrion\033[0m, \033[1;36mIoT-Agent\033[0m, \033[1;31mKeyrock\033[0m, \033[1;31mWilma\033[0m
        echo -e "- \033[1;34mOrion\033[0m is the context broker"
        echo -e "- \033[1;36mIoT-Agent\033[0m is configured for the UltraLight Protocol"
        echo -e "- \033[1;31mKeyrock\033[0m is an Identity Management Front-End"
        echo -e "- \033[1;31mWilma\033[0m is a PEP Proxy for Orion"
        startContainers orion.yml
        waitForKeyrock
        displayServices
        echo -e "Now open \033[4mhttp://localhost:3000\033[0m"
        ;;
    "stop")
        stoppingContainers
        ;;
    "create")
        echo "Pulling Docker images"
        docker-compose --log-level ERROR -f docker-compose/orion.yml -p fiware pull
        ;;
    *)
        echo "Command not Found."
        echo "usage: services [create|orion|iot-agent|stop]"
        exit 127;
        ;;
esac
```

Ilustración 54: Script de servicio para la puesta en marcha del sistema

Estos servicios hacen uso de las siguientes funciones:

- waitForKeyRock: Se encarga de hacer healthchecks[27] de Keyrock para comprobar su funcionamiento y muestra la respuesta por pantalla.

```
waitForKeyrock () {
    printf "⌛ Waiting for \033[1;31mKeyrock\033[0m to be available"

    while [ `curl -s -o /dev/null -w %{http_code} 'http://localhost:3005/version'` -eq 000 ]
    do
        echo -e "Keyrock HTTP state: " `curl -s -o /dev/null -w %{http_code} 'http://localhost:3005/version'` " (waiting for 200)"
        sleep 5
    done
    echo -e " \033[1;32mdone\033[0m"
}
```

Ilustración 55: Función waitForKeyrock

[26] Descarga de las imágenes necesarias a partir de un registro.
[27] Chequeos de funcionamiento. Normalmente se hacen peticiones sencillas que esperan una respuesta concreta para verificar el correcto funcionamiento del recurso.

- displayServices: Se encarga de mostrar los contenedores que están ejecutándose en Keyrock una vez se ha finalizado el arranque de todos los componentes, para comprobar que todos están funcionando.

```
displayServices () {
    echo ""
    docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}" --filter name=fiware-*
    echo ""
}
```

Ilustración 56: Función displayServices

- startContainers: Se encarga de levantar todos los componentes del sistema alojados en el archivo orion.yml. Para ello ejecuta un comando de docker-compose.

```
startContainers () {
    echo ""
    docker-compose --log-level ERROR -f docker-compose/orion.yml -p fiware up -d --remove-orphans
    echo ""
}
```

Ilustración 57: Función startContainers

- stoppingContainers: Se encarga de detener todos los contenedores definidos en el archivo orion.yml.

```
stoppingContainers () {
    echo "Stopping containers"
    docker-compose --log-level ERROR -f docker-compose/orion.yml  -p fiware down -v --remove-orphans
}
```

Ilustración 58: Función stoppingContainers

Archivo orion.yml

En este archivo es dónde se llevan a cabo todas las configuraciones de cada uno de los contenedores que conponen el sistema. Se procede a detallar cada uno de ellos.

Orion Context Broker

```
version: "3.5"
services:
  # Orion is the context broker
  orion:
    image: fiware/orion:2.2.0
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      default:
        ipv4_address: 172.18.1.9
    expose:
      - "1026"
    ports:
      - "1026:1026"
    command: -dbhost mongo-db -logLevel DEBUG
    healthcheck:
      test: curl --fail -s http://orion:1026/version || exit 1
```

Ilustración 59: Contenedor Orion en archivo orion.xml

En esta configuración, Orion Context Broker se establece en el puerto 1026. Este será el puerto con el que tendremos que establecer la comunicación para enviar o solicitar datos al context broker.

Este contenedor depende del contenedor mongo-DB.

Se asigna una dirección IP ficticia dentro de la red que conecta cada uno de los contenedores. Esta red se definirá más tarde.

Agente IOT Ultralight

```
# IoT-Agent is configured for the UltraLight Protocol
iot-agent:
  image: fiware/iotagent-ul:1.9.0
  hostname: iot-agent
  container_name: fiware-iot-agent
  depends_on:
    - mongo-db
    - orion
  networks:
    - default
  ports:
    - "4041:4041"
    - "7896:7896"
  environment:
    - IOTA_CB_HOST=orion # name of the context broker to update context
    - IOTA_CB_PORT=1026 # port the context broker listens on to update context
    - IOTA_NORTH_PORT=4041
    - IOTA_REGISTRY_TYPE=mongodb #Whether to hold IoT device info in memory or in a database
    - IOTA_LOG_LEVEL=DEBUG # The log level of the IoT Agent
    - IOTA_TIMESTAMP=true # Supply timestamp information with each measurement
    - IOTA_CB_NGSI_VERSION=v2 # use NGSIv2 when sending updates for active attributes
    - IOTA_AUTOCAST=true # Ensure Ultralight number values are read as numbers not strings
    - IOTA_MONGO_HOST=mongo-db # The host name of MongoDB
    - IOTA_MONGO_PORT=27017 # The port mongoDB is listening on
    - IOTA_MONGO_DB=iotagentul # The name of the database used in mongoDB
    - IOTA_HTTP_PORT=7896 # The port used for device traffic over HTTP
    - IOTA_PROVIDER_URL=http://iot-agent:4041
  healthcheck:
    test: curl --fail -s http://iot-agent:4041/iot/about || exit 1
```

Ilustración 60: Contenedor IoT-Agent en el archivo orion.yml

Este contenedor depende del context broker Orion y de la base de datos mongoDB.

Se observa que este contenedor cuenta con dos puertos, lo cuál cumple lo explicado en el apartado 4.2.2.

Por último se observan diversas variables de entorno, de las cuales procedemos a comentar las más interesantes:

- IOTA_CD_HOST=orion y IOTA_CD_PORT=1026. Establece a Orion Context Broker como el destino al que actualizar su contexto.
- IOTA_NORT_PORT=4041 y IOTA_HHTP_PORT=7896. Con estas variables se establecen los puertos norte y sur, cada uno de ellos encargado de una tarea concreta.
- IOTA_PROVIDER_URL=hhtp://iot-agent:4041. Establece una URL con la que establecer la comunicación.

Keyrock Identity Management

```
# Keyrock is an Identity Management Front-End
keyrock:
  image: fiware/idm:7.7.0
  container_name: fiware-keyrock
  hostname: keyrock
  networks:
    default:
      ipv4_address: 172.18.1.5
  depends_on:
    - mysql-db
  ports:
    - "3005:3005"
  environment:
    - DEBUG=idm:*
    - IDM_DB_HOST=mysql-db
    - IDM_DB_PASS_FILE=/run/secrets/my_secret_data
    - IDM_DB_USER=root
    - IDM_HOST=http://localhost:3005
    - IDM_PORT=3005
    - IDM_HTTPS_ENABLED=${IDM_HTTPS_ENABLED}
    - IDM_HTTPS_PORT=3443
    - IDM_ADMIN_USER=javier
    - IDM_ADMIN_EMAIL=javier@test.com
    - IDM_ADMIN_PASS=test
  secrets:
    - my_secret_data
  healthcheck:
    test: curl --fail -s http://localhost:3005/version || exit 1
```

Ilustración 61: Contenedor Keyrock en el archivo orion.yml

Este componente depende exclusivamente de la base de datos my-sql, donde se almacena toda la información necesaria para el correcto funcionamiento de Keyrock.

Se establece una dirección IP ficticia de nuevo para poder comunicarse con los diferentes componentes del sistema.

Se procede a comentar las variables de entornos más interesantes:

- IDM_HOST: http://localhost:3005 : Se establece URL para comunicarnos con Keyrock, ya sea a través de su API o a través de la interfaz gráfica que proporciona.
- IDM_HTTPS_ENABLED = ${IDM_HTTPS_ENABLED}. Activa o desactiva el soporte para el protocolo HTTPS.
- IDM_ADMIN_USER=javier, IDM_ADMIN_EMAIL = javier@test.com, IDM_ADMIN_PASS=test. Con estas variables creamos un usuario administrador en el momento de la instalación del sistema, para poder gestionar Keyrock desde el principio.

Wilma PEP Proxy

```
# PEP Proxy for Orion
orion-proxy:
  image: fiware/pep-proxy:7.7.0
  container_name: fiware-orion-proxy
  hostname: orion-proxy
  networks:
    default:
      ipv4_address: 172.18.1.10
  depends_on:
    - keyrock
  ports:
    - "1027:1027"
  expose:
    - "1027"
  environment:
    - PEP_PROXY_APP_HOST=orion
    - PEP_PROXY_APP_PORT=1026
    - PEP_PROXY_PORT=1027
    - PEP_PROXY_IDM_HOST=keyrock
    - PEP_PROXY_HTTPS_ENABLED=false
    - PEP_PROXY_AUTH_ENABLED=false
    - PEP_PROXY_IDM_SSL_ENABLED=false
    - PEP_PROXY_IDM_PORT=3005
    - PEP_PROXY_APP_ID=app-dckr-site-0000-xpresswebapp
    - PEP_PROXY_USERNAME=pep_proxy_00000000-0000-0000-0000-000000000000
    - PEP_PASSWORD=test
    - PEP_PROXY_PDP=idm
    - PEP_PROXY_MAGIC_KEY=1234
```

Ilustración 62: Contenedor de Proxy PEP Wilma en el archivo orion.yml

Este componente depende únicamente de keyrock, el cuál es el PDP del sistema y Wilma el componente PEP, como ya se explicó en el apartado 5.1.1

Se establece una dirección IP para poder comunicarse con los diferentes componentes del sistema.

Se procede a comentar algunas de las variables más interesantes:

- PEP_PROXY_IDM_HOST=keyrock. Se establece como IDM por defecto a Keyrock.
- PEP_PROXY_APP_ID=app-dckr-site.0000-xpresswebapp. Es el identificador único de una aplicación concreta, a la cuál Wilma le provee de seguridad.

mongoDB, mySQL y network

```
  # Databases
  mongo-db:
    image: mongo:3.6
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "27017"
    ports:
      - "27017:27017"
    networks:
      - default
    command: --bind_ip_all --smallfiles
    volumes:
      - mongo-db:/data

  mysql-db:
    restart: always
    image: mysql:5.7
    hostname: mysql-db
    container_name: db-mysql
    expose:
      - "3306"
    ports:
      - "3306:3306"
    networks:
      default:
        ipv4_address: 172.18.1.6
    environment:
      - "MYSQL_ROOT_PASSWORD_FILE=/run/secrets/my_secret_data"
      - "MYSQL_ROOT_HOST=172.18.1.5"
    volumes:
      - mysql-db:/var/lib/mysql
      - ../mysql-data:/docker-entrypoint-initdb.d/:ro
    secrets:
      - my_secret_data
networks:
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24
```

Estos son los contenedores que alojan a los diferentes sistemas de bases de datos, cuya funcionalidad ya se detalló en el apartado 3.5.

Puesta en marcha del Sistema

Para llevar a cabo el proceso de puesta en marcha, se deben seguir los siguientes pasos:

1. Ejecutar el siguiente comando para hacer pull de los elementos necesarios:

```
$ sudo ./services create
```

2. Ejecutar el comando encargado de la puesta en marcha del sistema:

```
$ sudo ./services orion
```

Finalmente, comprobamos que el sistema se ha desplegado correctamente, observando el mensaje final que muestra por pantalla el comando anterior.

```
javier@ubuntu:~/app/tutorials.PEP-Proxy$ sudo ./services orion
[sudo] password for javier:
Stopping containers
Stopping fiware-orion-proxy ... done
Stopping fiware-keyrock     ... done
Stopping db-mongo           ... done
Stopping db-mysql           ... done
Removing fiware-orion-proxy ... done
Removing fiware-keyrock     ... done
Removing db-mongo           ... done
Removing db-mysql           ... done
Starting six containers Orion, IoT-Agent, Keyrock, Wilma MongoDB and MySQL databases.
- Orion is the context broker
- IoT-Agent is configured for the UltraLight Protocol
- Keyrock is an Identity Management Front-End
- Wilma is a PEP Proxy for Orion

Creating db-mongo ... done
Creating db-mysql     ... done
Creating fiware-orion   ... done
Creating fiware-keyrock   ... done
Creating fiware-iot-agent   ... done
Creating fiware-orion-proxy ... done
Creating fiware-tutorial    ... done

⌛ Waiting for Keyrock to be availableKeyrock HTTP state:  000  (waiting for 200)
Keyrock HTTP state:  000  (waiting for 200)
Keyrock HTTP state:  000  (waiting for 200)
Keyrock HTTP state:  000  (waiting for 200)
Keyrock HTTP state:  000  (waiting for 200)
Keyrock HTTP state:  000  (waiting for 200)
 done

NAMES               STATUS                          PORTS
fiware-tutorial     Up 38 seconds (healthy)         0.0.0.0:3000-3001->3000-3001/tcp
fiware-orion-proxy  Up 47 seconds                   0.0.0.0:1027->1027/tcp
fiware-iot-agent    Up 48 seconds (healthy)         0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp, 4061/tcp
fiware-keyrock      Up 52 seconds (health: starting)  3000/tcp, 0.0.0.0:3005->3005/tcp
fiware-orion        Up 54 seconds (healthy)         0.0.0.0:1026->1026/tcp
```

Ilustración 63: Resultado de la ejecución del comando sudo ./services orion

A partir de este momento, ya se puede utilizar el sistema, siempre y cuando se haya ejecutado el programa gps.py en la Raspberry Pi.

ANEXO C: COMANDOS CURL

B.1 Ilustración 18

```
curl -X POST \
  http://localhost:1026/v2/entities/ \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: fbc7074b-9305-473a-af2e-26c4383f62fa' \
  -H 'cache-control: no-cache' \
  -d '{
    "id": "urn:ngsi-ld:Store:001",
    "type": "Store",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Antonio Montes 17",
            "addressRegion": "Alcalá de Guadaira",
            "addressLocality": "Sevilla",
            "postalCode": "41500"
        },
        "metadata":{
"verified":{
"value":true,
"type": "Boolean"

}

}
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [37.336702, -5.845826]
        }
    },
    "name": {
        "type": "Text",
        "value": "Alcalá de Guadaira"
    }
}'
```

B.2 Ilustración 34

```
curl -X POST \
  http://localhost:4041/iot/services \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 3065d7d5-3948-425f-9333-b61edd112f43' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
    "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
    "cbroker":     "http://orion:1026",
    "entity_type": "MOVE",
    "resource":    "/iot/d"
   }
 ]
}'
```

B.3 Ilustración 35

```
curl -X POST \
  http://localhost:4041/iot/devices \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 06bd435b-2244-405c-ab4b-39024483b7e1' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
   {
    "device_id":   "Move01",
    "entity_name": "urn:ngsi-ld:Move01",
    "entity_type": "Move",
    "timezone":    "Europe/Madrid",
    "attributes": [
      { "object_id": "lat", "name":"latitud", "type":"Float"},
      { "object_id": "lon", "name":"longitud", "type":"Float"},
      { "object_id": "alt", "name":"altitud", "type":"Float"},
      { "object_id": "vel", "name":"velocidad", "type":"Float"},
      { "object_id": "dir", "name":"direccion", "type":"Float"},
      { "object_id": "sub", "name":"subida", "type":"Float"},
      { "object_id": "fec", "name":"fecha", "type":"String"}
     ]
   }
 ]
}
'
```

B.4 Ilustración 38

```
curl -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: e4cd9775-d32d-46a8-b523-438dc35234e0' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

B.5 Ilustración 41

```
curl -X POST \
  http://localhost:3005/oauth2/token \
  -H 'Accept: application/json' \
  -H 'Authorization: Basic dHV0b3JpYWwtZGNrci1zaXRlLTAwMDAteHByZXNzd2ViYXBwOnR1dG9yaWFsLWRja3Itc2l0ZS0wMDAwLWNsaWVudHNlY3JldA==' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H 'Postman-Token: 57957c5e-667c-4228-a53d-ab68134bca8c' \
  -H 'cache-control: no-cache'
```

B.6 Ilustración 44

```
curl -X GET \
  'http://localhost:1027/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: 53d98258-3761-4d8d-984d-afb620a5a5ee' \
  -H 'X-Auth-Token: bec6bbc45655af2cc340b911b9e391f92b9bf7a4' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

B.7 Ilustración 46

```
curl -X GET \
  'http://localhost:1027/v2/entities/urn:ngsi-ld:Move01?options=keyValues' \
  -H 'Postman-Token: 53d98258-3761-4d8d-984d-afb620a5a5ee' \
  -H 'X-Auth-Token: ****************************' \
  -H 'cache-control: no-cache' \
  -H 'fiware-service: TFG' \
  -H 'fiware-servicepath: /'
```

ANEXO D: ARCHIVOS ADJUNTOS

En el CD adjunto se encuentran todos los archivos necesarios para reproducir el comportamiento del presente proyecto.

Se procede a detallar el contenido:

/Archivos_adjuntos

/Docker

/docker-compose

orion.yml

/mysql-data (vacío)

secrets.txt

services

/Postman

TFG.postman_collection.json (Peticiones utilizadas en Postman)

/Raspberry

gps.py

REFERENCIAS

[1] Luis Martinez Ruiz, Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware, Sevilla 2018.

[2] [En línea] Available: https://www.blog.andaluciaesdigital.es/que-es-fiware-y-por-que-es-clave-para-el-emprendimiento/

[3] [En línea] Available: https://www.fiware.org/developers/

[4] [En línea] Available: https://fiware-idm.readthedocs.io/en/latest/#identity-manager-keyrock

[5] [En línea] Available: https://naylampmechatronics.com/sensores-posicion-inerciales-gps/106-modulo-gps.html

[6] [En línea] Available: https://www.electronicoscaldas.com/es/boards-raspberry-pi/800-raspberry-pi-1-modelo-b-plus.html

[8] [En línea] Available: https://www.pcbox.com/productos/ask3557/portatil-asus-f550-x798h-core-i7-3537u

[9] [En línea] Available: https://www.paradigmadigital.com/dev/postman-gestiona-construye-tus-apis-rapidamente/

[10] [En línea] Available: https://code.tutsplus.com/es/tutorials/using-the-requests-module-in-python--cms-28204

[11] [En línea] Available: https://www.redhat.com/es/topics/containers/what-is-docker

[12] [En línea] Available: https://docs.docker.com/compose/

[13] [En línea] Available: https://fiware-orion.readthedocs.io/en/1.8.0/index.html

[14] [En línea] Available: https://fiware-academy.readthedocs.io/en/latest/security/wilma/index.html

[15] [En línea] Available: https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

[16] [En línea] Available: https://neoattack.com/neowiki/mysql/

[17] [En línea] Available: https://www.fiware.org/about-us/

[18] [En línea] Available: https://www.fiware.org/developers/catalogue/

[19] [En línea] Available: https://fiware-orion.readthedocs.io/en/master/index.html

[20] [En línea] Available: https://pt.slideshare.net/FI-WARE/orion-context-broker-webminar/4

[21] [En línea] Available: https://fiware-sth-comet.readthedocs.io/en/latest/

[22] [En línea] Available: https://fiware-cygnus.readthedocs.io/en/latest/

[23] [En línea] Available: https://flume.apache.org/

[24] [En línea] Available: https://fiware-iotagent-json.readthedocs.io/en/latest/index.html

[25] [En línea] Available: https://ricveal.com/blog/primeros-pasos-mqtt/

[26] [En línea] Available: https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html#user-programmers-manual

[27] [En línea] Available: https://wirecloud.readthedocs.io/en/stable/

[28] [En línea] Available: https://knowage.readthedocs.io/en/latest/

[29] [En línea] Available: https://kurento.readthedocs.io/en/stable/

[30] [En línea] Available: http://localhost:3005/login

[31][Enlínea] Available:

https://fiware-idm.readthedocs.io/en/latest/user_and_programmers_guide/user_guide/index.html

[32][Enlínea]Available:
https://fiwareidm.readthedocs.io/en/latest/user_and_programmers_guide/application_guide/index.html

[33] [En línea] Available: https://www.slideshare.net/FI-WARE/fiware-wednesday-webinars-how-to-secure-fiware-architectures

[34] [En línea] Available: https://es.slideshare.net/rOzacC/xacml-37071981

[35] [En línea] Available: https://solidgeargroup.com/oauth2-protocolo-de-autorizacion?lang=es